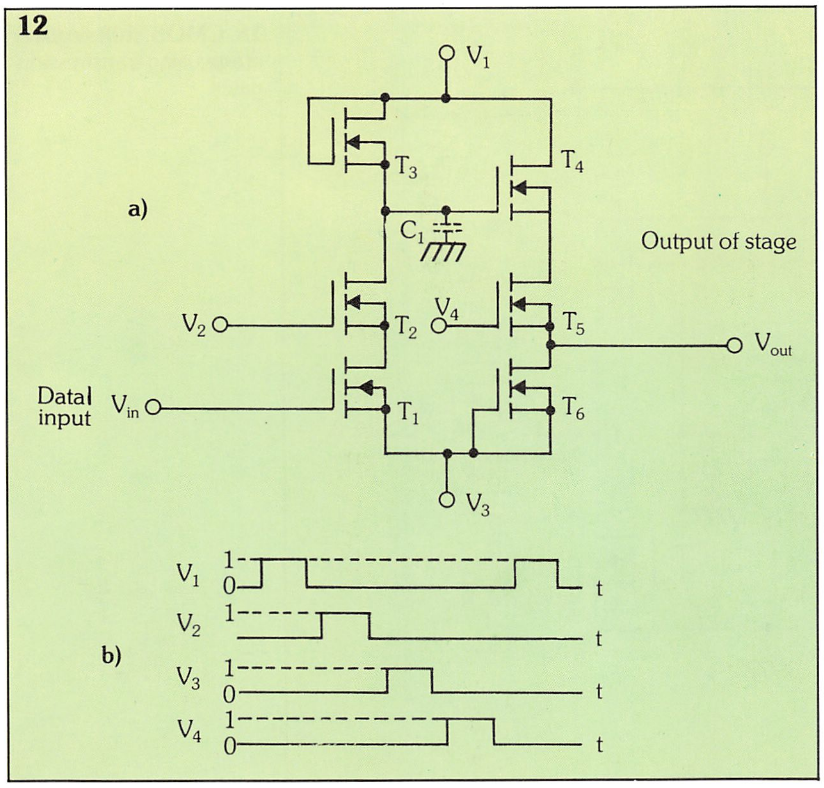
(continued from part 18)

Four-phase stages

In the earlier examples of MOSFET shift registers, the parasitic capacitances of transistors within each register stage were used to store charge. The stored charge was then discharged to another parasitic capacitor of another transistor. The disadvantage of this arrangement is that it takes time to charge and discharge capacitors. *Figure 12a* shows an arrangement which uses only one capacitor, and therefore operates more quickly, but it requires a four-phase clock signal in order to work.

Examining *figure 12a*, we see that when clock signal V_1 is at logic 1, capacitor C_1 charges to this value and remains charged even when signal V_1 returns to logic 0. When clock signal V_2 goes to logic 1, and there is a logic 1 data input, transistors T_1 and T_2 turn on and allow capacitor C_1 to discharge through transistors T_1 and T_2. In this way, the complement of the stage input is stored by capacitor C_1. In a similar manner, the complement of the value at C_1 is transferred to the stage output. A timing diagram of the four-phase clock signal is shown in *figure 12b*.

12. (a) Stage of a four-phase clock shift register; **(b)** its timing diagram.



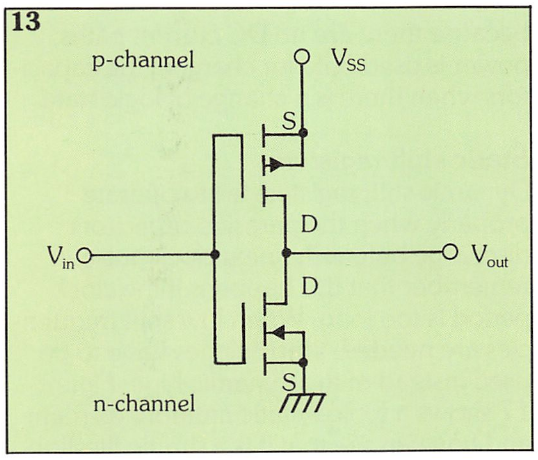
CMOS shift registers

CMOS registers can also be made using the inverter arrangements we saw earlier. A CMOS inverter is shown in *figure 13* and this can be coupled with an NMOS electronic switch (*figure 14*) to produce the same kinds of circuits we have previously seen. However, use of this single transistor switch causes a low noise margin. This is because the voltage between gate and source, which must be present to turn the MOSFET on (the threshold voltage) is about 2 V. So if logic 1 (i.e. 10 V) is applied to the gate, the highest source voltage must be 8 V for the transistor to operate. The output capacitor C_1, can therefore only charge to a maximum of about 8 V.

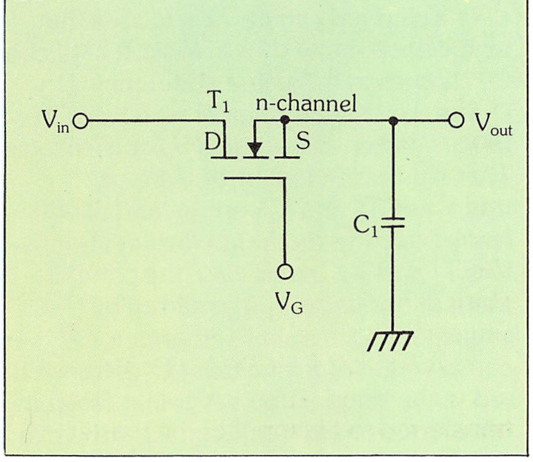
To increase the noise margin, switches with two complementary transistors can be used which allow the output capacitor to charge to the maximum logic 1 level, i.e. 10 V. We'll examine how this is so by looking at the CMOS switch of *figure 15*. As the transistors are complementary (one n-channel and one p-channel), the gate voltages V_1 and V_2 are of opposite polarity to make both devices switch on or off simultaneously. We saw a similar switch in *Digital Electronics 15* (called a transmission gate) used to form a three-state output stage for CMOS logic devices.

When gate voltage V_1 is logic 1 and gate voltage V_2 is 0, a 10 V logic 1 input V_{in}, causes transistor T_1 to be *off* because the gate-source voltage is less than the transistor's threshold voltage ($V_1 - V_{in} = 10 - 10 = 0$ V). However, the gate-source voltage of transistor T_2 is higher than the

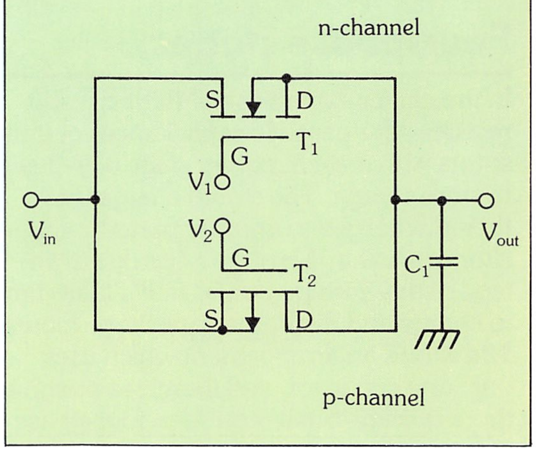
13. CMOS inverter.



14



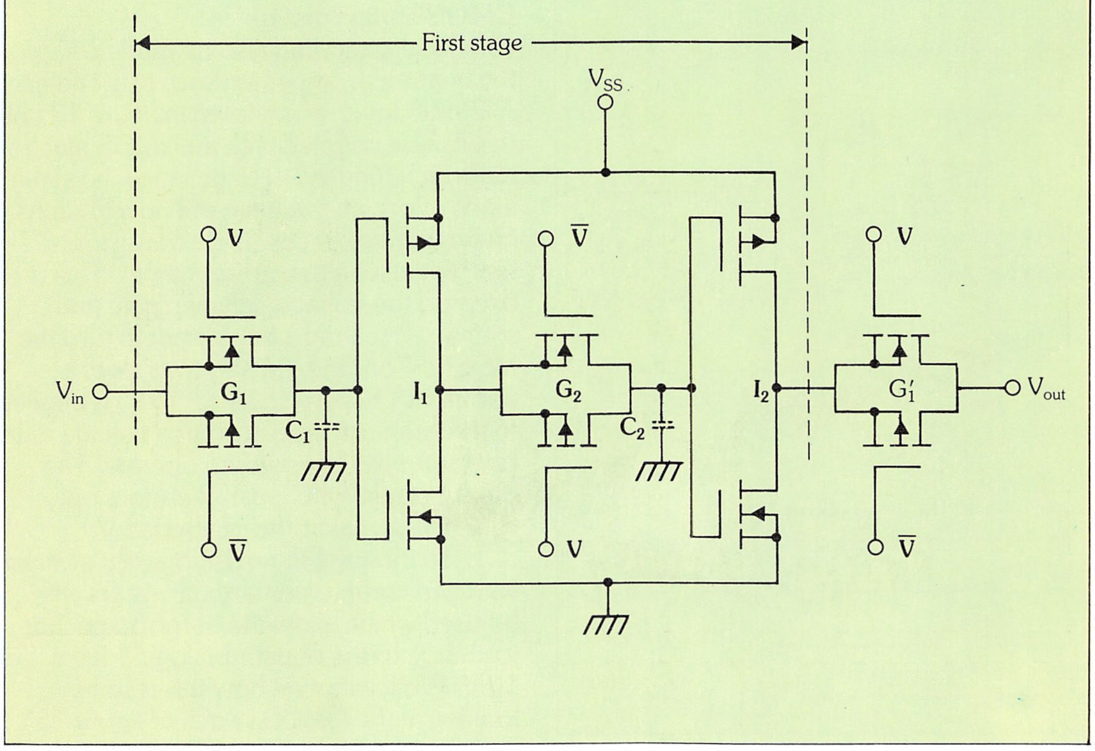
15



14. NMOS electronic switch.

15. CMOS switch – note that V_1 and V_2 are of opposite polarity to make both devices switch on or off simultaneously.

16



16. CMOS shift register stage using transmission gates.

transistor's threshold voltage ($V_2 - V_{in} = 0 - 10 = -10$ V) so transistor T_2 is on and capacitor C_1 charges to 10 V.

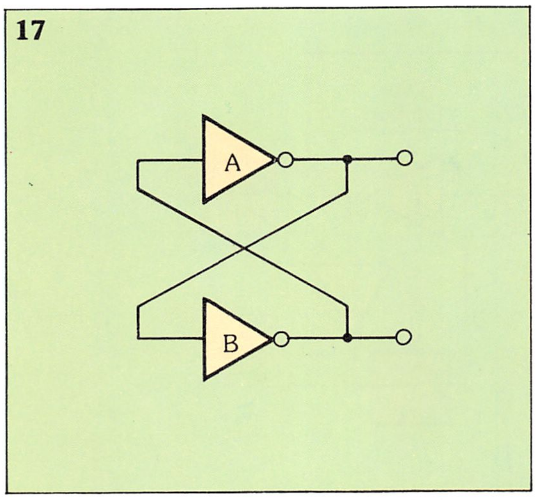
If the input goes to logic 0 when gate voltage V_1 is 1 and V_2 is 0, transistor T_2 is now off because its gate-source voltage is less than the threshold voltage. However, transistor T_1 is now on, allowing capacitor C_1 to discharge.

Figure 16 illustrates a CMOS shift register stage using transmission gates which behaves in the same way as the dynamic MOS register of *figure 11*. The power dissipation of this circuit is very low because there are no DC current paths; power is used only for charging the capacitors when there is a change of logic state.

Static shift registers

Dynamic shift registers fail to operate properly when the parasitic capacitors discharge before the next clock pulse – remember that this happens if the clock period is too long. When low shift frequencies are needed, **static** stages have to be used instead of the dynamic type. *Figure 17* shows a typical static memory element and you can see that it is a simple flip-flop

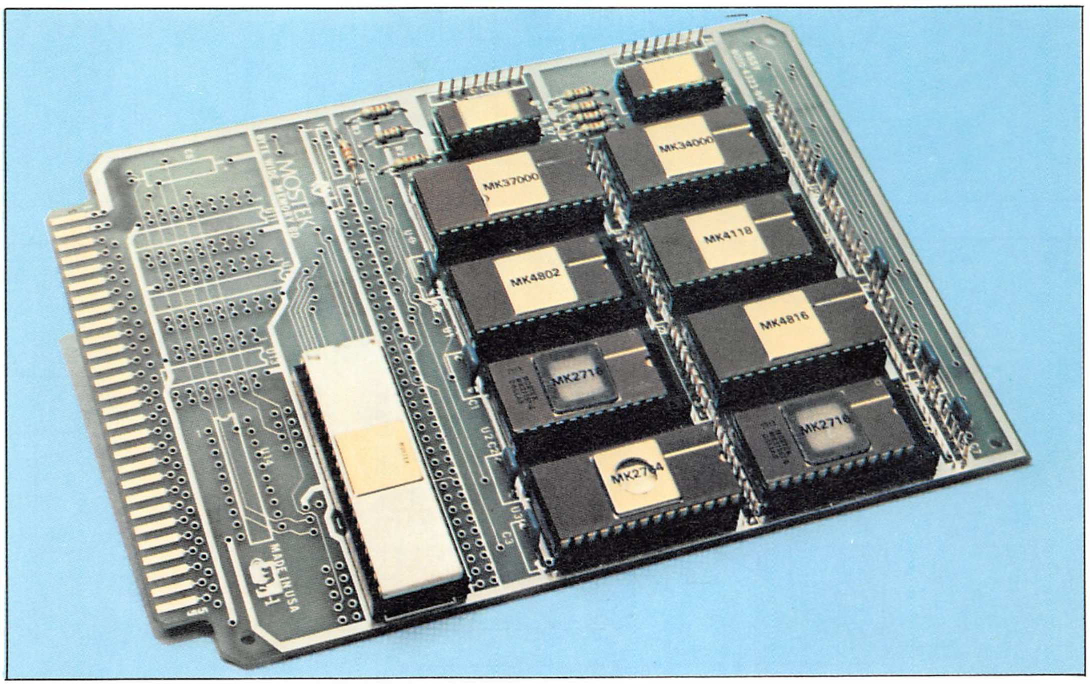
17. A typical static memory element: a simple flip-flop made from two cross-coupled inverters.



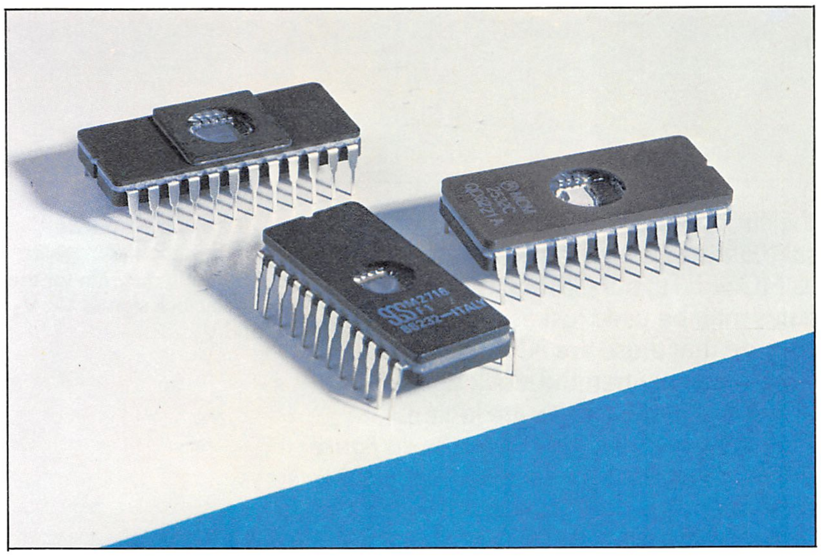
made from two cross coupled inverters. At the moment this circuit is useless because all the inverters do is hold themselves in one state or the other, without any means of changing the logic state. In *figure 18a* a block diagram arrangement of a shift register stage is shown which uses master and slave flip-flops, together with the necessary switches to control the flip-flop states. In *figure 18b* the complementary clock pulses which are needed to drive the stage are shown. Let's look at how it works.

When clock signal C, is logic 1, switch 1 is on and data at the stage input is transferred to the input of the master

Right: memory expansion board with ROM (MK 37000 and 34000), RAM (MK 4802, 4118, 4816) and EPROM (MK 2716, 2764). (Photo: Mostek).



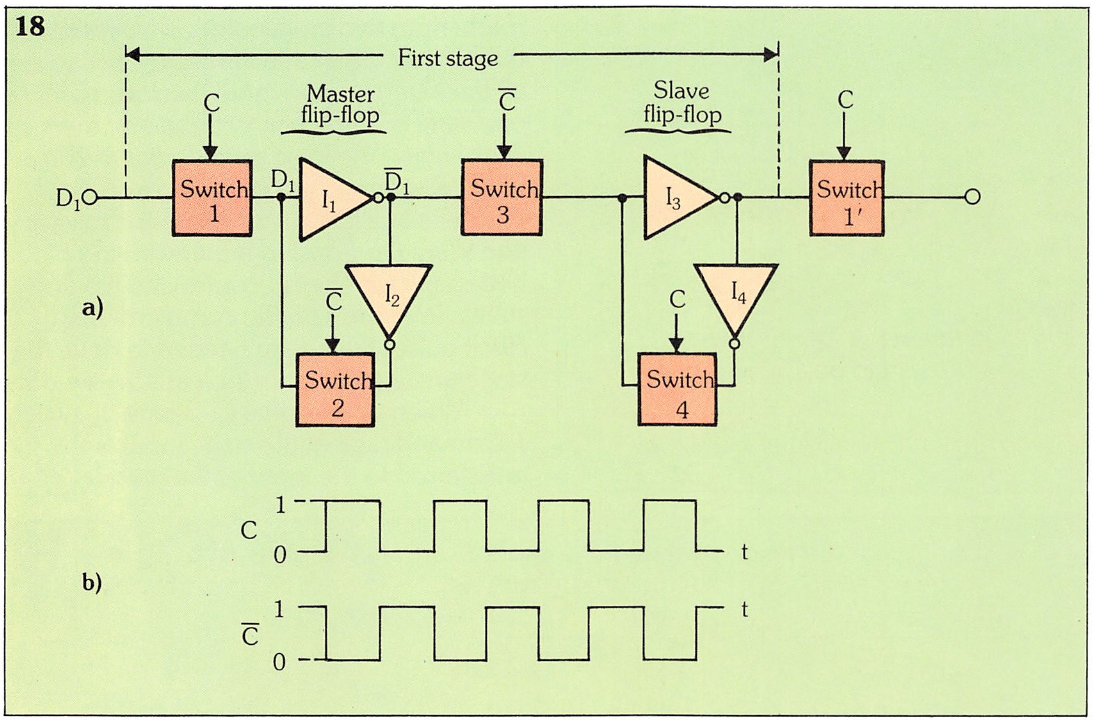
Below: EPROM memory chips. (Photo SGS).



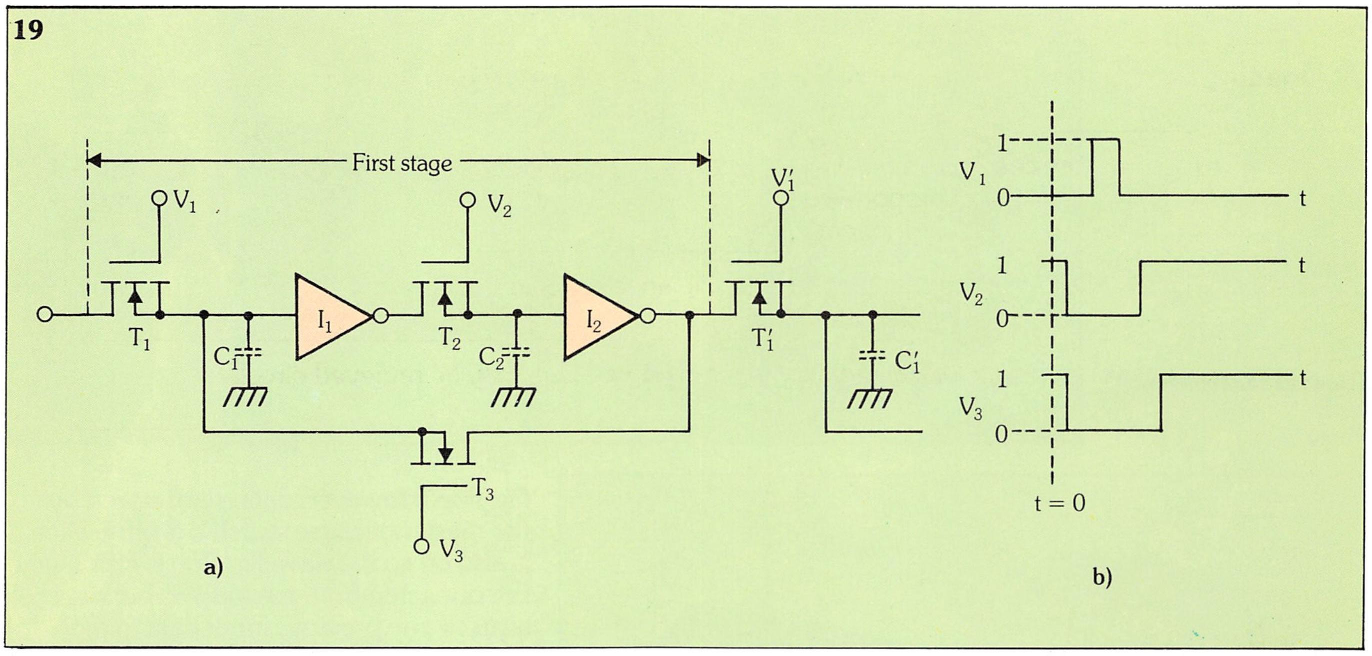
flip-flop. However, data is not stored by the master because switch 2 is off. Switch 3 is also off so the slave flip-flop is effectively disconnected from the master, but switch 4 is on so any previous input data to the slave is stored and its complement is present at the stage output.

When the clock signal changes state, the master flip-flop stores the data at its input because switch 2 is now on. The complement of this data is present at the master flip-flop output and, as switch 3 is now on, is applied to the slave input. We can see that each flip-flop stores data on one half-cycle of the clock signal but transfers data on the other half-cycle.

Since there is no reliance on the



18. (a) First stage of a shift register using master and slave flip-flops; **(b)** the complementary clock pulses needed to drive this stage.



19. (a) Three-phase clock static shift register; **(b)** timing diagram for the three clock signals V_1, V_2 and V_3.

parasitic capacitance of the switches, the clock frequency is not crucial, as in previous cases. In fact, even if the clock signal is stopped for a while the information is still preserved.

Three-phase static shift register stage
The static shift register stage of *figure 18* consists of four switches and four inverters. This number of components may be reduced by using a three-phase clock and the circuit shown in *figure 19a*. For the sake of simplicity, the switches are shown as MOSFETs, but CMOS transmission gates may be used just as easily. As before, assume that these are NMOS devices, which are on when the clock signals presented to their gates are logic 1.

Now we'll see how it works. In *figure 19b* a timing diagram for the three clock signals V_1, V_2 and V_3 is shown. When t = 0, clock signals V_2 and V_3 are at logic 1, but clock signal V_1 is logic 0. Therefore, the loop including the inverters I_1 and I_2 forms a flip-flop.

Transistors T_1 and T_1' are off, so the flip-flop is isolated from other stages. When clock signals V_2 and V_3 go to logic 0, the data in the flip-flop is stored on parasitic capacitances C_1 and C_2. Before these capacitances discharge, clock signal V_1 goes to logic 1. This allows data to be passed to the next stage through transistor T_1' and new data to be applied across capacitor C_1 through transistor T_1. Clock signal V_1 returns to logic 0 and the new data is stored across capacitor C_1. Before the capacitor discharges, clock signal V_2 goes to logic 1, turning transistor T_2 on, passing the complement of the new data to capacitor C_2. Finally, clock signal V_3 goes to logic 1, turning on transistor T_3 recreating a flip-flop and storing the new data in static form. As we can see, the order in which the transistors turn on and off is very important.

Glossary

clock phases	collection of clock signals, used to control separate parts of a digital circuit, which are out of phase with each other, i.e. occurring at different but related times
dynamic storage	data storage method, relying on the voltages stored across parasitic capacitances within digital devices. Data is lost, i.e. the parasitic capacitances discharge, if the data is not updated at frequent intervals (approximately 1 μs)
loading control	a control input of a shift register which instructs the register to recirculate data at its output back to the input
parasitic capacitances	capacitances caused by the physical attributes of an electronic component. Such capacitances may or may not affect operation of the component
parasitic resistances	resistances caused by an electronic component's physical make-up (see parasitic capacitors)
random access memory (RAM)	digital memory in which stored data may be retrieved directly
recirculation path	a path from a shift register's output back to its input, which allows data to continuously recirculate around the register
refresh	term used to describe the operation of updating data stored in a dynamic digital memory
serial access memory	digital memory in which stored information must be retrieved in a certain manner, usually in the same order in which it was originally written
static storage	digital memory in which data can be stored for indefinite periods without the need for refreshing

11 BASIC COMPUTER SCIENCE

Resource management

Resource managers

In *Basic Computer Science 10* we discussed the development and role of the operating system. One of its functions, that of management of the computer system's resources, we shall now go on to look at in greater detail.

Resource management involves the monitoring and control of the various system resources required by an applications program as it is being executed. These resources are of two types: hardware, such as the CPU, memory and I/O peripherals; and software, such as the utilities programs (compiler/interpreter/assembler, loader program and data file manipulation programs).

In batch processing systems, these separate resources can be allocated to the control of one of five **resource managers** – these are individual modules that form part of the set of programs comprising the operating system. (It is important to note that the labels given here for the various resource managers will not be the same for all manufacturers' operating systems, they form an example from one operating system only. In fact, even the term resource manager does not have universal currency: the term monitor may be used or there may be no separate labels for individual functions.)

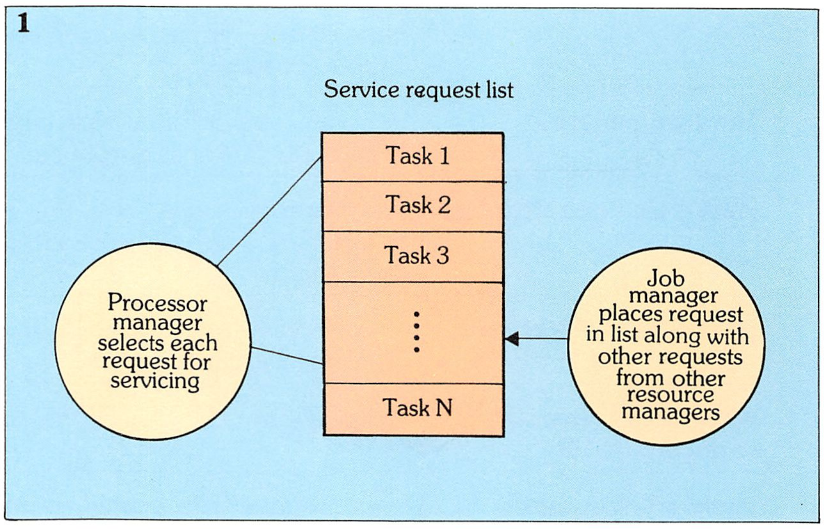
The **job manager** reads in the program or job and determines which resources are needed. It then ensures that these resources are available, schedules the job for execution, monitors the job throughout its execution and releases the resources when the job has finished; the **memory manager** assigns and releases memory as needed; the **processor manager** is responsible for processor time; the **I/O manager** assigns I/O operations to specific devices; and the **file manager** handles requests for the storage and retrieval of data files.

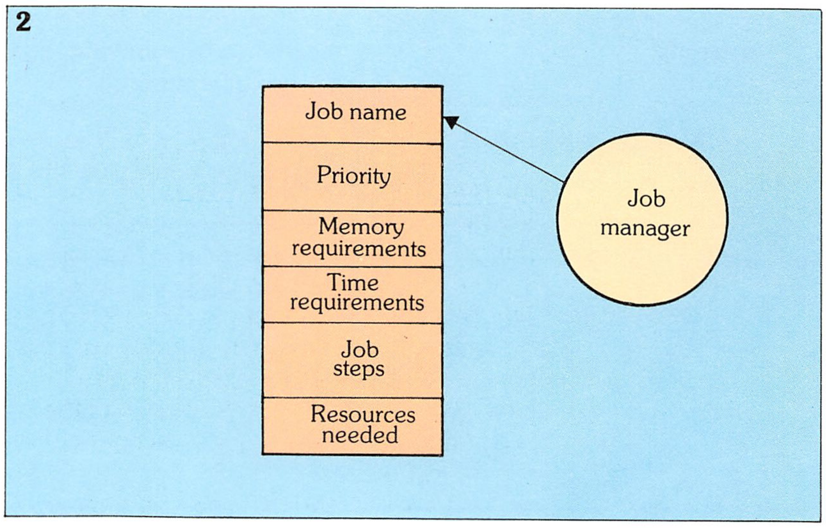
The functions of these modules may seem familiar – they are similar to the functions of the resident monitor discussed for early operating systems in *Basic Computer Science 10*.

Each resource manager has a number of lists associated with it and it is via these lists that resource managers request servicing from one another. For example, should the job manager require information concerning the availability of an I/O

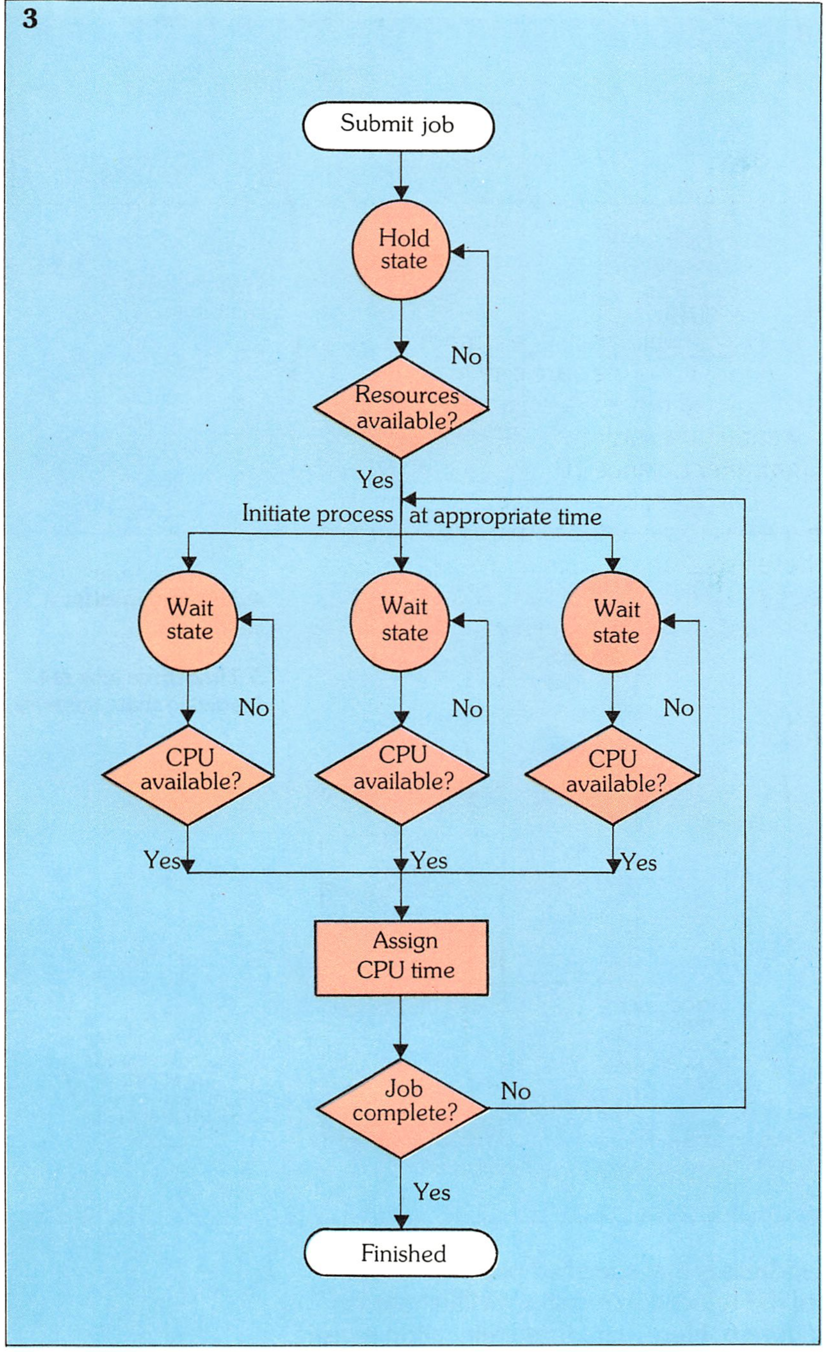
1. Processor manager's service request list.

2. Job control block.


1
Service request list
Task 1
Task 2
Task 3
Task N
Processor manager selects each request for servicing
Job manager places request in list along with other requests from other resource managers



3



3. Flow diagram for the overall job execution sequence.

peripheral, then a request is placed in the I/O manager's **service request list**. The I/O manager then examines the list and performs the appropriate processing to service the request. Similarly, should a function of the job manager itself need to be serviced (remember, resource managers are themselves programs that require execution) then a request is placed in the processor manager's request list as shown in *figure 1*.

The job is read in by the job manager which records all resource requirements. Information regarding the name of the job, its priority, its memory and time requirements, the data file needed and the request for the compiler, for example, is contained in the job control statements (these were discussed in *Basic Computer Science 10*).

The job manager then requests the necessary space on the disk from the information manager. The information manager checks the **file directory** to see if there is space available; this directory is held on disk and contains a list of all files, their location and size. A file name is given to each of the new jobs. As each job is read and filed onto disk, a **job control block** is generated: this is simply an internal table used by the job manager for executing each job (*figure 2*).

The job manager examines the control blocks and selects the job with the highest priority; requests are then made to the memory manager, the I/O manager and the information manager ensuring that all the resources needed are available. The job is then placed in a **wait** state in central memory by the memory manager, and attached to the service request list of the processor manager; the job is now ready to be processed by the CPU.

The processor manager examines this list and assigns a certain amount of processing time to each task; at the end of this time period, an interrupt is made to the processor, the task entry is deleted, and the CPU is assigned to the next task on its list. When the last process on the list has been serviced, the processor manager returns to the top of the list – this loop is repeated until there are no further requests. This method of sharing time between tasks is known as a **round-robin arrangement**. The processor manager itself also requires CPU time in order to be executed – this is done inbetween the tasks on the list, after each time interrupt.

When the application program has been executed, the resources that have been used are released. Any output from the program is not usually sent directly to the I/O device specified, but **spooled** to the disk. This means that the output is placed in a buffer until either the printer is free or until the output for other jobs is ready to be printed. (The term derives from early computing days when output was held on spools of magnetic tape.) The overall job sequence is illustrated in *figure 3*.

The memory manager

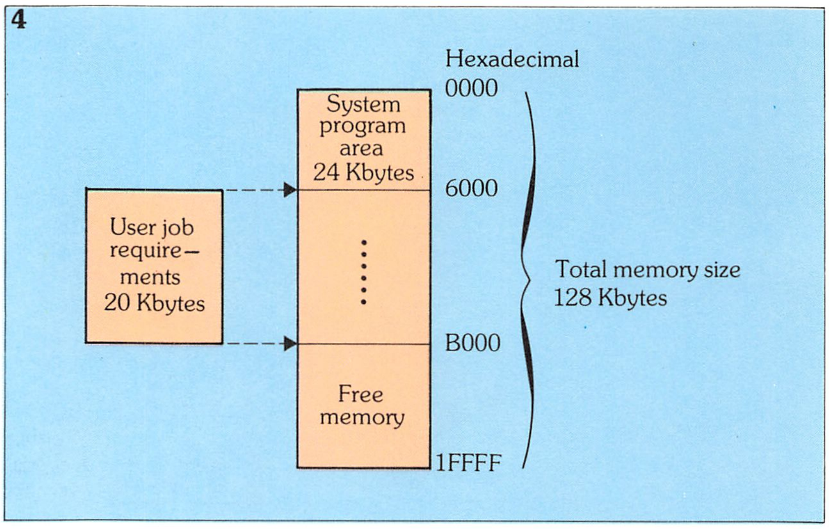
The memory manager, as we have said, is responsible for assigning memory to specific tasks. If only one job is being run on the system, then memory control is relatively simple. In *figure 4*, a memory map for a single job is shown. Of a total memory size of 128 Kbytes, 24 Kbytes are used for the operating system programs and 20 Kbytes are needed for the user's job.

Figure 5 illustrates how three jobs are loaded to share memory. As long as memory is available, jobs are simply loaded into adjacent locations. However, suppose that jobs A and C are finished, but that B is not; job D is awaiting execution – it is larger than either memory slot vacated by jobs A or C, but not larger than the space available if they were added together. Job D, therefore, cannot be executed until B is either finished or is moved to another location. Moving job B would present difficulties if it was in progress because all memory references would need to be changed.

One solution to this problem is the use of **relocation registers**; each job is assembled or compiled as if it started at memory location zero. The *actual* memory address is that specified plus the contents of the relocation register, as illustrated in *figure 6*. Here, with three relocation registers, job B can be moved and its relocation register changed to indicate its new position; job D can then be loaded. Although job movement is eased by this method, each job requires its own register.

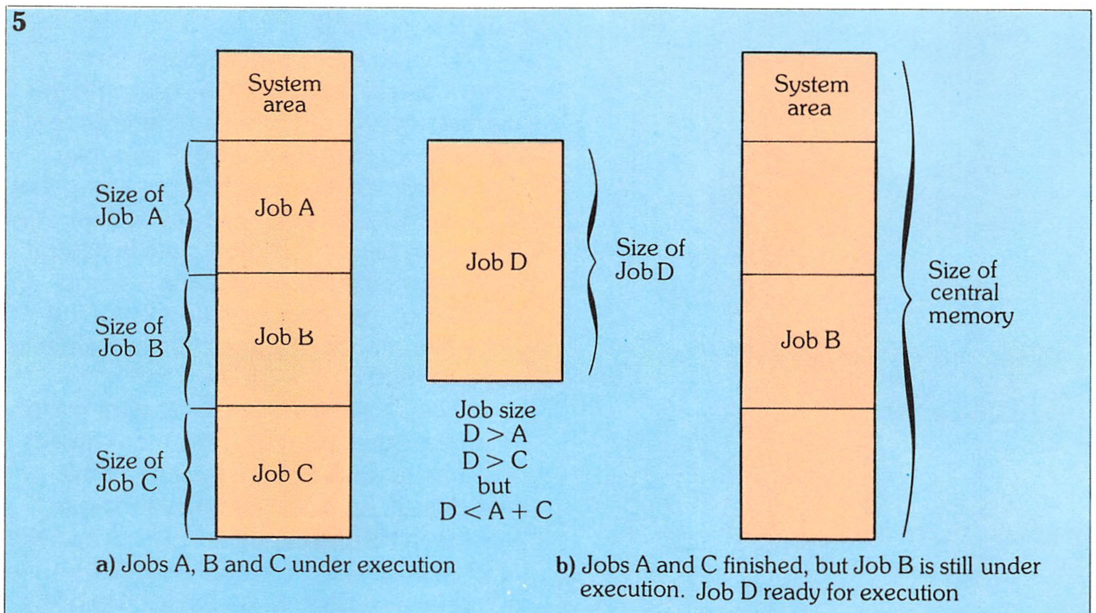
A more efficient method of handling job movement is to use the **fixed page system** (this was discussed in *Basic Computer Science 10*). In *figure 7*, job A is divided into pages, each of, say, 2 Kbytes. The available address space may be much greater than the available memory and as each job is loaded, the memory manager

4



Hexadecimal
0000
System program area 24 Kbytes
6000
User job requirements 20 Kbytes
B000
Free memory
1FFFF
Total memory size 128 Kbytes

5

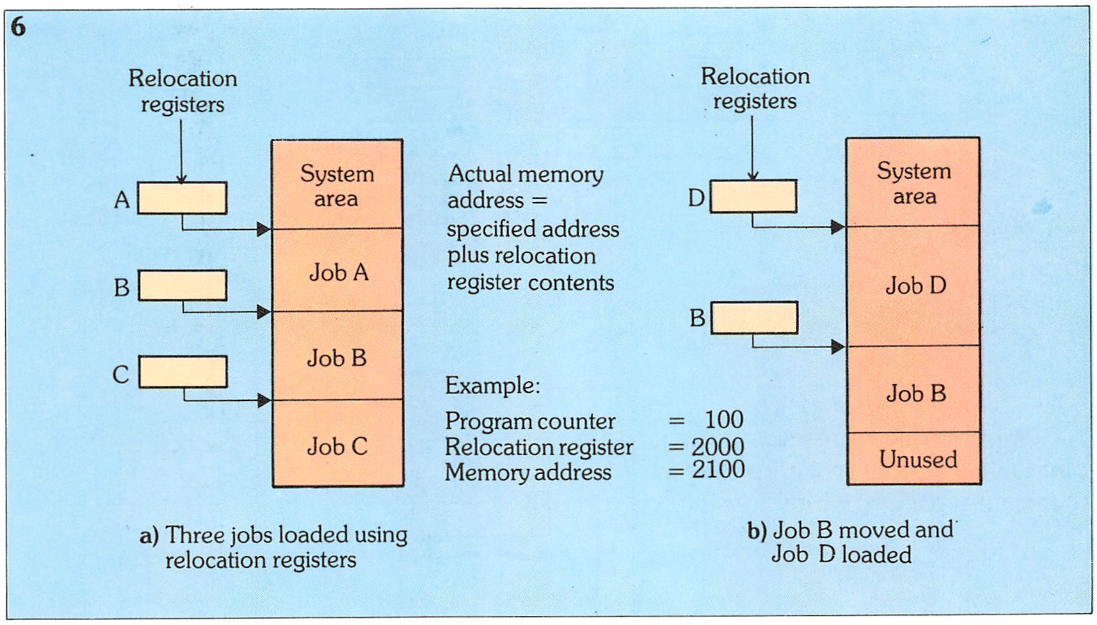


System area
Size of Job A
Job A
Size of Job B
Job B
Size of Job C
Job C
Job D
Size of Job D
Job size D > A D > C but D < A + C
System area
Job B
Size of central memory
a) Jobs A, B and C under execution
b) Jobs A and C finished, but Job B is still under execution. Job D ready for execution

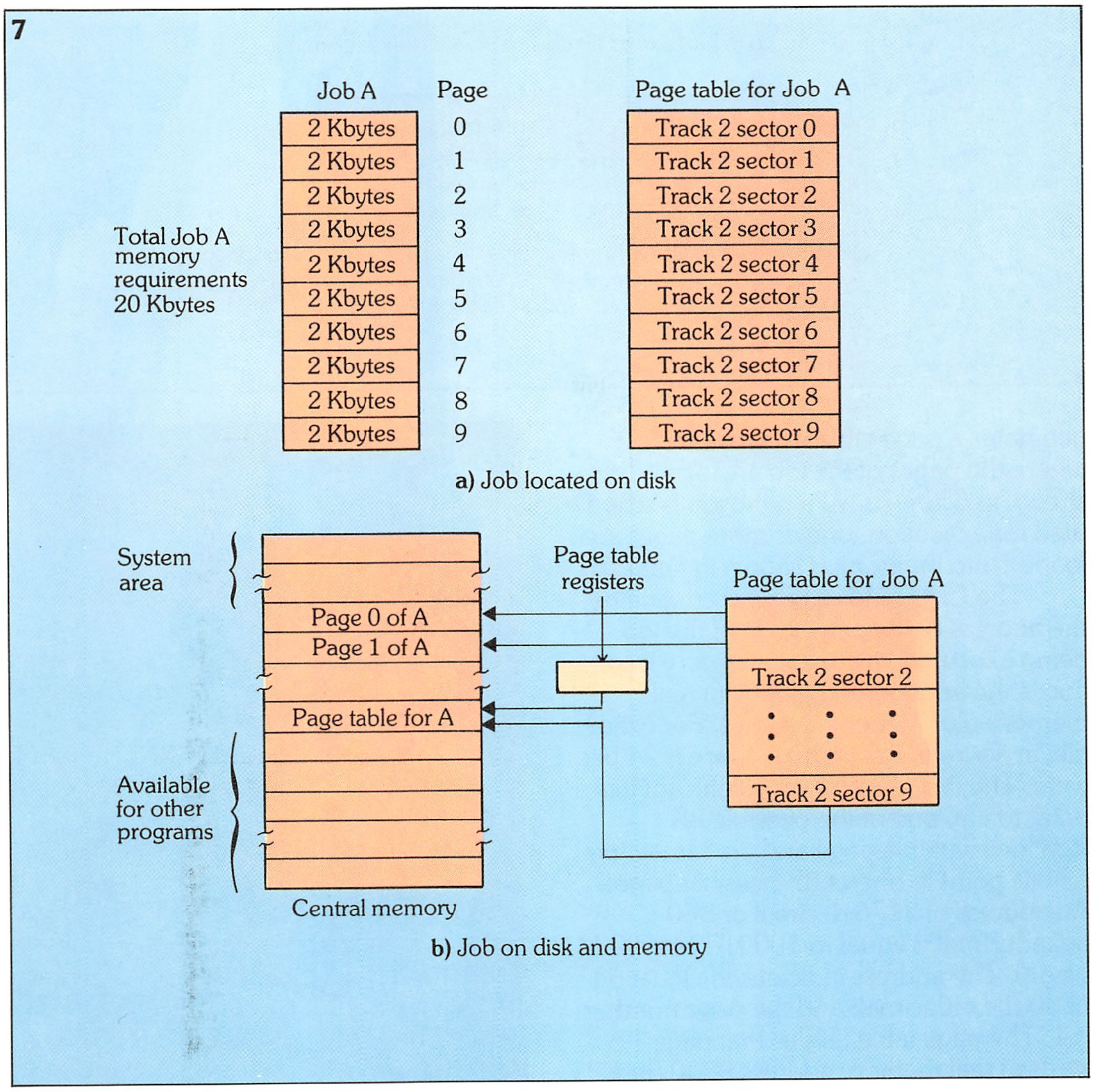
4. Memory map for a single job.

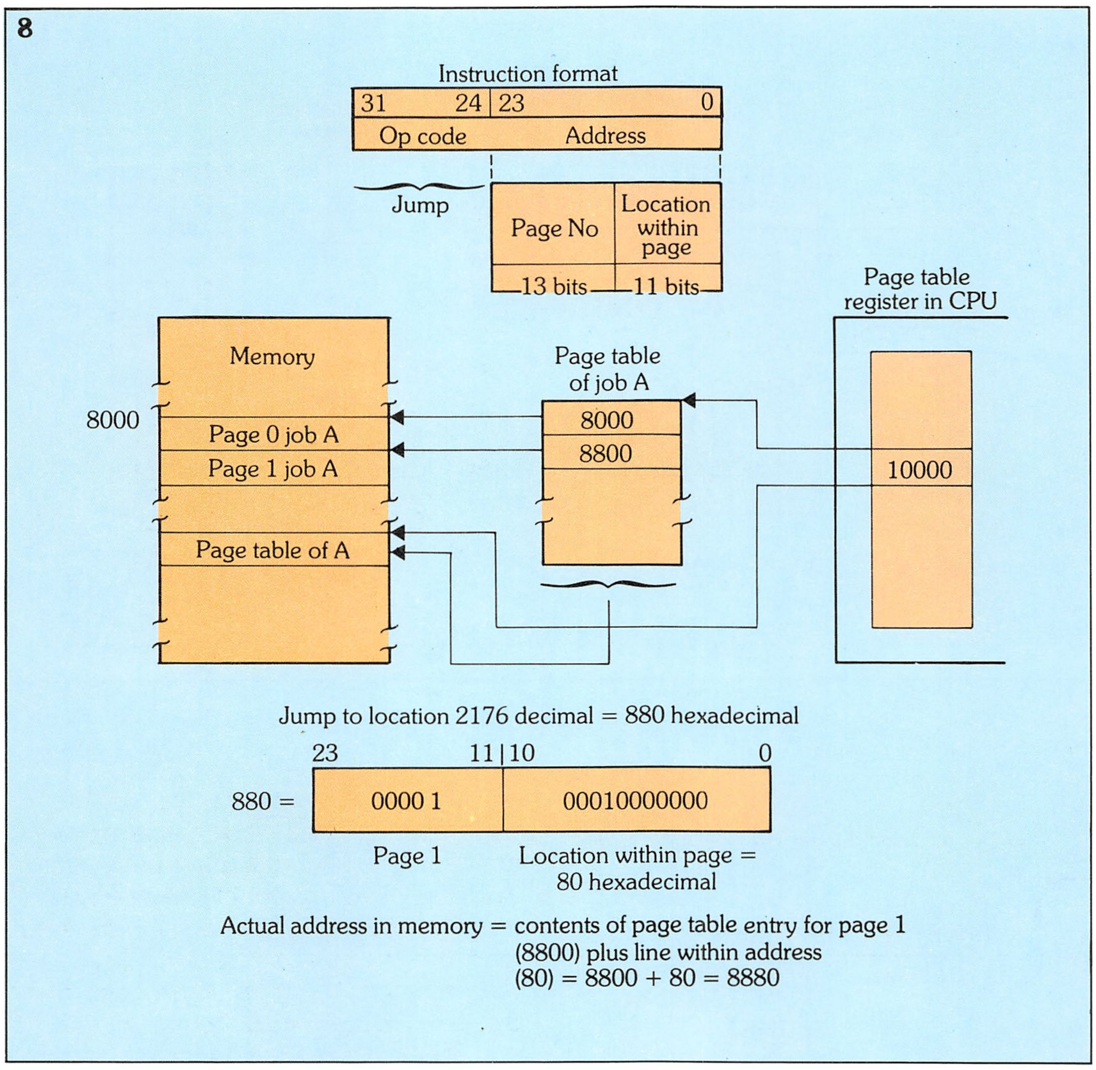
5. How three jobs are loaded to share memory.

6. Loading jobs into memory using relocation registers.


6
Relocation registers
A
B
C
System area
Job A
Job B
Job C
Actual memory address = specified address plus relocation register contents
Example:
Program counter = 100
Relocation register = 2000
Memory address = 2100
a) Three jobs loaded using relocation registers
Relocation registers
D
B
System area
Job D
Job B
Unused
b) Job B moved and Job D loaded

7. Handling job movement in memory using the fixed page system.


7
Job A
Page
Page table for Job A
2 Kbytes 0 Track 2 sector 0
2 Kbytes 1 Track 2 sector 1
2 Kbytes 2 Track 2 sector 2
2 Kbytes 3 Track 2 sector 3
2 Kbytes 4 Track 2 sector 4
2 Kbytes 5 Track 2 sector 5
2 Kbytes 6 Track 2 sector 6
2 Kbytes 7 Track 2 sector 7
2 Kbytes 8 Track 2 sector 8
2 Kbytes 9 Track 2 sector 9
Total Job A memory requirements 20 Kbytes
a) Job located on disk
System area
Page 0 of A
Page 1 of A
Page table for A
Available for other programs
Central memory
Page table registers
Page table for Job A
Track 2 sector 2
Track 2 sector 9
b) Job on disk and memory


8
Instruction format
31
24
23
0
Op code
Address
Jump
Page No
Location within page
13 bits
11 bits
Page table register in CPU
Memory
Page table of job A
8000
Page 0 job A
Page 1 job A
8000
8800
10000
Page table of A
Jump to location 2176 decimal = 880 hexadecimal
23
11
10
0
880 =
0000 1
00010000000
Page 1
Location within page = 80 hexadecimal
Actual address in memory = contents of page table entry for page 1 (8800) plus line within address (80) = 8800 + 80 = 8880

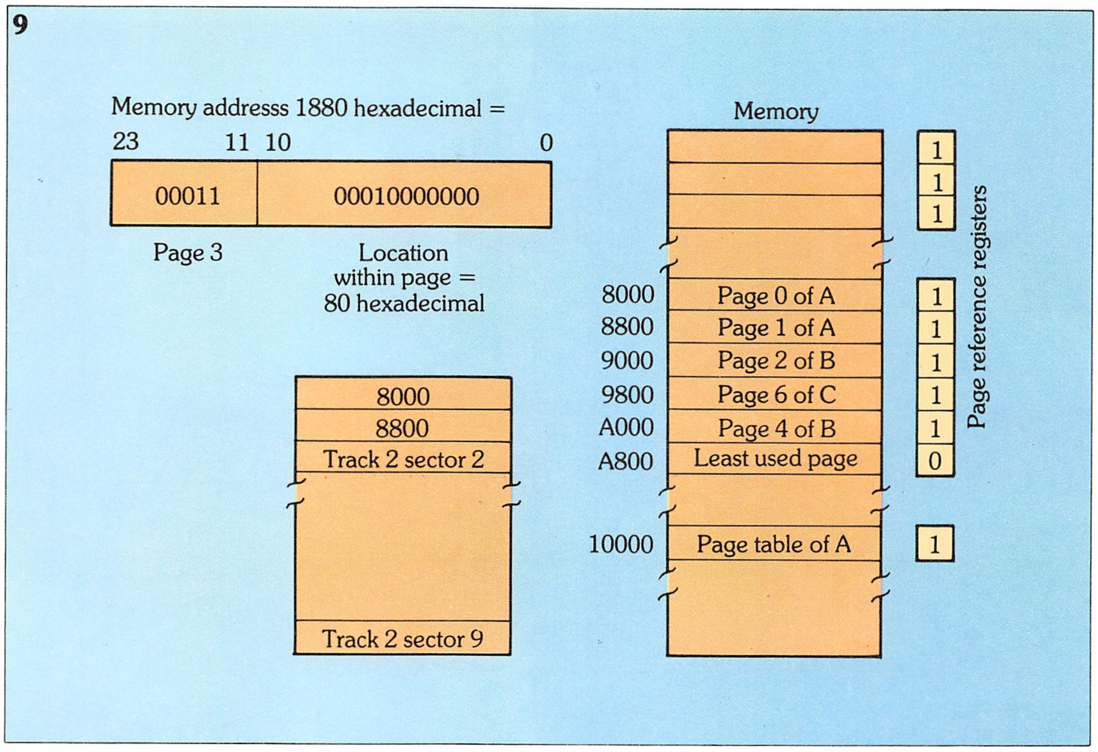
8. The actual memory location is the address of the page in real memory plus the page address.

Below: a terminal used in a hotel to book accommodation, compile accounts etc. (Photo: Philips).

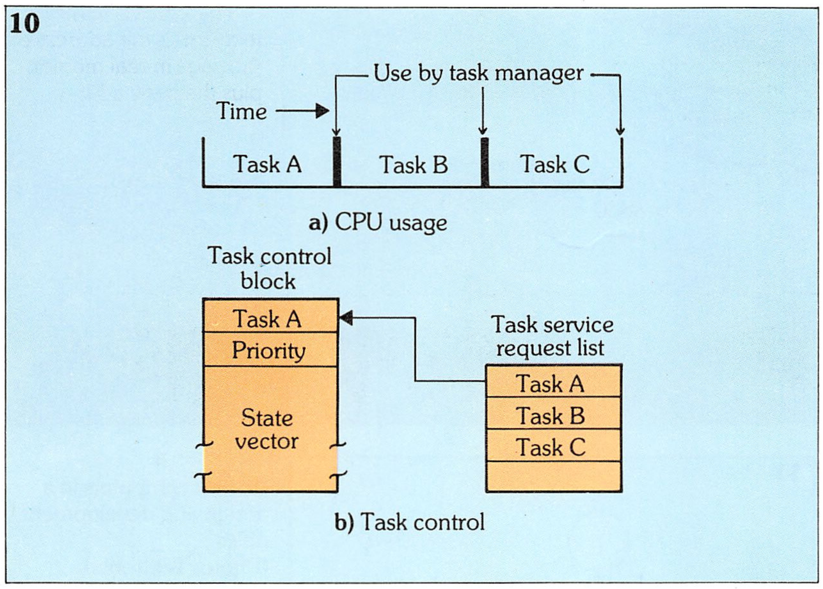
generates a page table which indicates where the pages of the job are located, as shown in *figure 7a*. When the job is scheduled for execution, one or more pages are loaded into memory as shown in *figure 7b*.

The CPU holds a register containing the address of the page table of the job being executed; the actual memory location is the address of the page in real memory plus the page address. For example, in *figure 8*, the address space is 24 bits wide. The first eleven least significant bits point to any one of the possible 2K locations within a page and the remaining 13 bits point to one of 8K possible pages. An address of 2176 decimal or 880 hexadecimal is equal to 100010000000 binary. The address indicates the location of 80 (hexadecimal) and the page number is 1. The page table tells us that page 1 starts at real memory address 8800 (hex-

9. How to determine the address of a page which is not in memory.



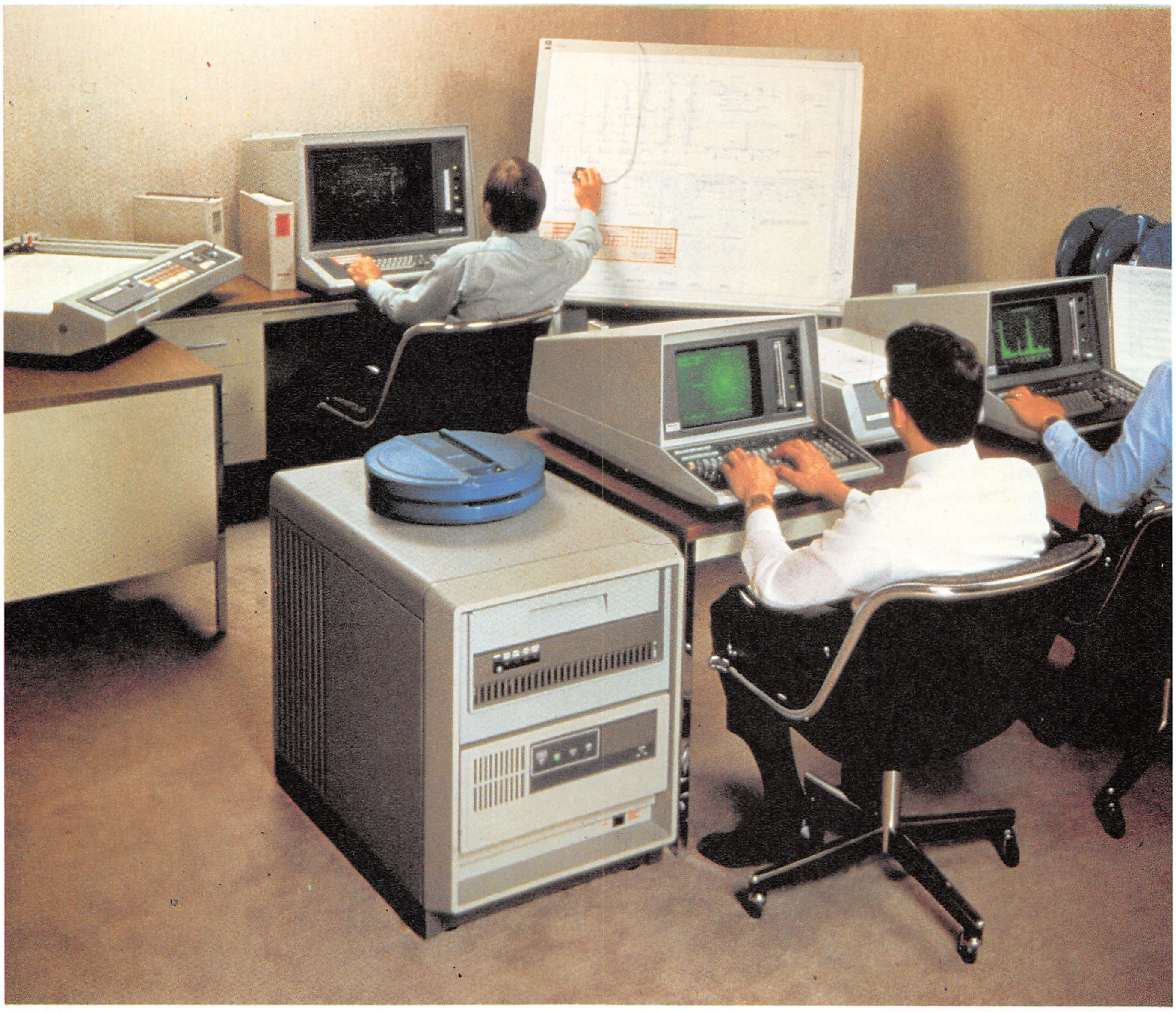
10. The processor manager examines its service request list, and assigns a task control block and a certain amount of CPU time to each task.



adecimal), and so the address 880 (hex) is translated as real memory location 8880.

If the program instructs the CPU to access a location not present in the pages which are currently in memory, a **page fault** occurs (this is a hardware function). The CPU processes the next task on the processor manager's service request list while that page is placed in memory. Many systems which control memory in this way have registers associated with each page in memory which keep track of the number of references made to each page; when a page fault occurs, these registers are examined. The pages that have been used the least, as shown by their registers, are transferred back to disk and new pages are loaded in their place – this is the least recently used (LRU) paging strategy, first discussed in the previous chapter.

The memory manager contains tables indicating which pages are free. *Figure 9* illustrates the process of determining the address of a page which is not in memory. A memory reference to 1880 (hex) is translated as location 80, page 3. The hardware then fetches location 10000 (which is the address contained in the current page table register from *figure 8*) plus 3 and finds that this location only exists on disk. A page fault has occurred, which is indicated by an interrupt. Control passes to the memory manager which examines the one bit registers associated with every page. It finds that page 21 was not used since the last examination of these registers. All the page reference registers are then usually set to zero by the memory manager, and the contents of page 21 are **paged out** to disk. The contents of disk location track 2 sector 2 is then **paged in** or loaded into memory locations A800 to AFFF. This memory



location is then fetched and execution continues.

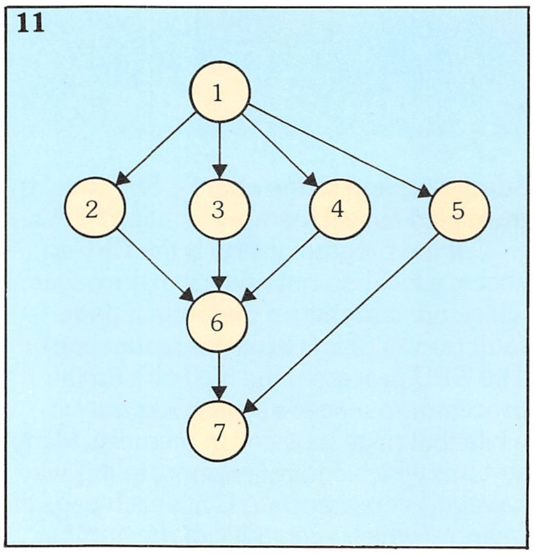
Some operating systems keep an account of the page reference numbers and if the reference was a read or a store; if no store occurred, then the contents of the memory will match that of the disk. The contents of that page in memory can then be simply overwritten.

Another place on disk, the **swap-space**, contains swapped out pages which *have* been altered.

The processor manager

In a multiprogramming system the processor manager examines its service request list (*figure 10*). A certain amount of time is

11



$$\begin{tikzcd}
 & & 1 \arrow[dll] \arrow[d] \arrow[dr] \arrow[drr] & & \\
2 \arrow[drr] & & 3 \arrow[d] & 4 \arrow[dl] & 5 \arrow[ddll] \\
 & & 6 \arrow[d] & & \\
 & & 7 & &
\end{tikzcd}$$

Above: computers in a design and development office. (Photo: Tektronix).

11. Processing sequence for a 7 task job.

assigned to each task and the hardware is set to generate an interrupt after each task 'time' has elapsed.

A task **control block** is assigned to each task on the list which is similar in function to the job control block already discussed. It contains such data as task name, priority, location, the addresses of all the tables used by the program, and the **state vector**. The state vector is the location where all the registers used by the task are stored when an interrupt occurs; when the CPU returns to the task which was interrupted, these register contents are restored. This state vector is hardware specific; there are other methods of storing registers.

When running a job, there are three main processes involved: compilation, loading and execution. As you can imagine, timing is particularly important – loading must not occur before compiling, for example. *Figure 11* illustrates a more complex situation: task 7 cannot be executed before 5 and 6 have finished since 7 is dependent on the results of both; similarly, task 6 cannot be run before tasks 2, 3, and 4; and finally, tasks 2, 3, 4 and 5 are all dependent on the result of task 1, so cannot be run until task 1 is complete.

The processor manager holds service requests for all tasks at the same time. How does it therefore control the processing in the correct sequence? The processor manager uses a procedure known as **flagging** to label each task in order to prevent its execution until all the required preceding tasks are complete. These flags, known as **global variables**, indicate the existence of a certain condition when **set** and the absence of that condition when **clear**.

For example, if a task must not be executed until N preceding tasks have been run, then the global variable, say P, is set to the value – N. As each of the preceding tasks is executed, P is incremented until it reaches zero. The processor manager checks the value of P after each task has been run: if P is not zero then another task on the service list is executed, and so on until P is cleared (i.e. set to zero).

The global variable in the example (*figure 11*) for task 6, P_6, is initially set to -3 and is incremented after each of tasks 2, 3 and 4 until it is zero (*figure 12*).

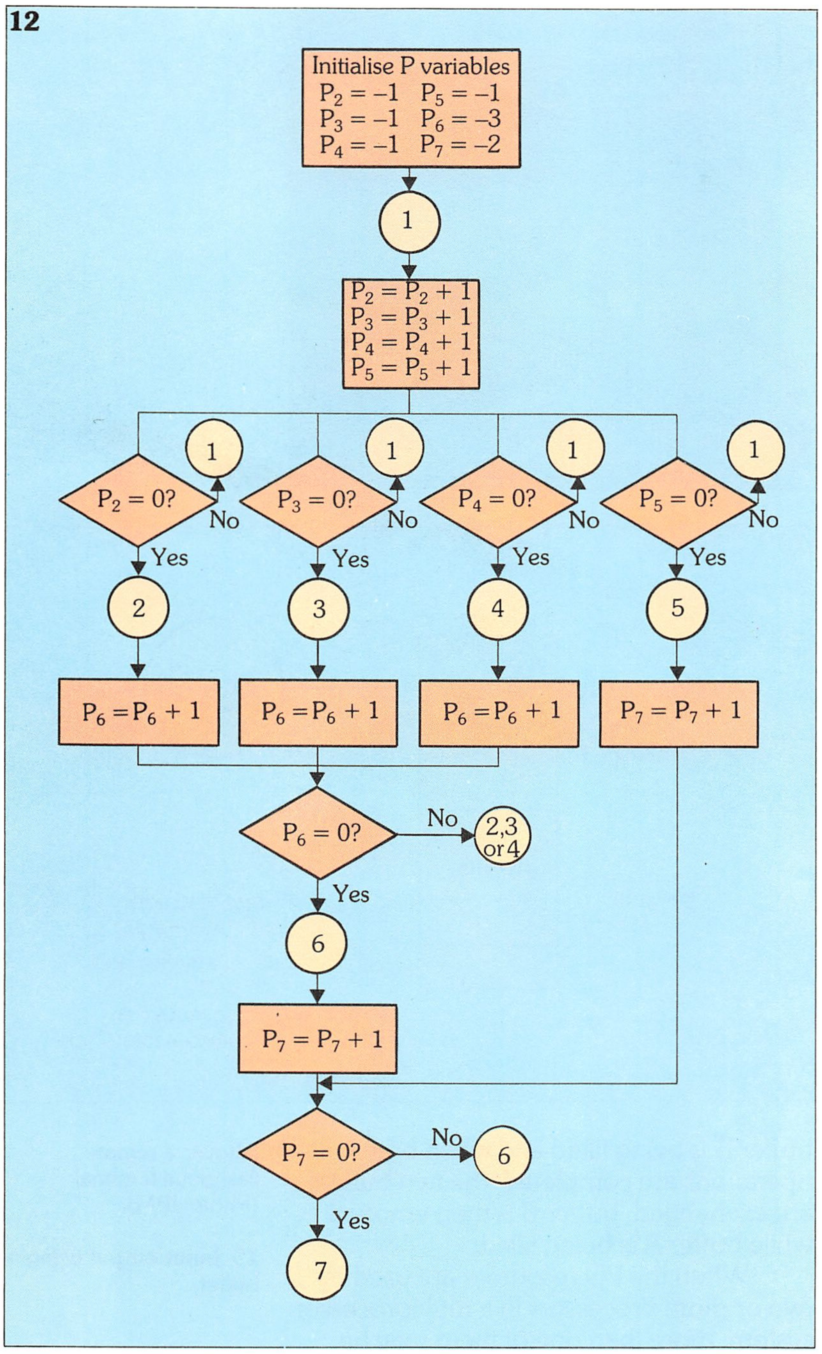
Multiprocessing describes a system where several independent CPUs operate simultaneously: each working on different but related tasks. This is a special case where the processor manager has a choice of processors, each sharing memory.

The I/O manager

The I/O manager is responsible for all input/output operations. The information manager locates the I/O device and the address of a data file that has been

12. Using global variables to control the processing sequence.

12



requested by a program, and passes this information to the I/O manager which then performs the input/output functions. An appropriate **channel status word** for a particular I/O operation is set up which directs the **data channel** or **I/O processor** to begin. The channel status word is a block of information which specifies: the external device that will send or receive; whether the transfer is an input or an output operation; the number of words to be transferred; the memory locations to which data is to be sent or from which it is to be obtained; and what to do when the I/O is finished.

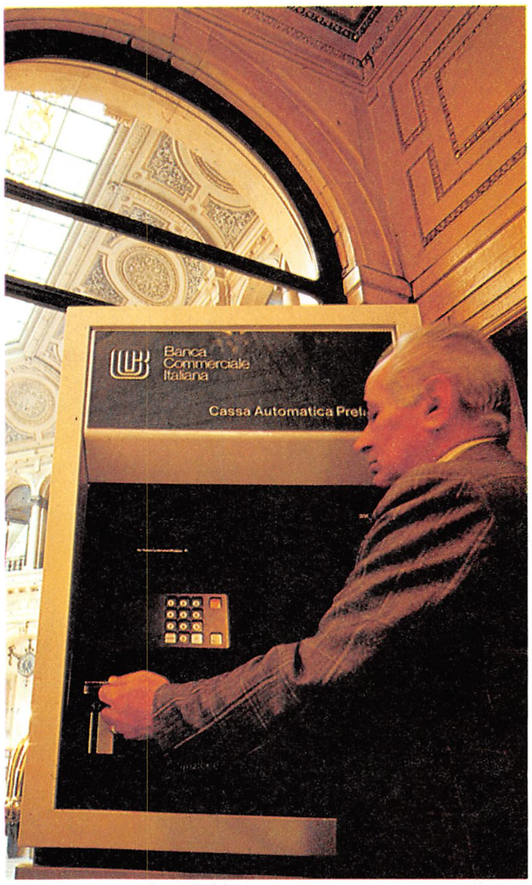
If a request is made to a device which is busy, the I/O processor sets up a queue of requests; when the device is free, requests are serviced on a first in, first out basis. The I/O manager maintains a table of all I/O resources showing the address, status, logical unit assigned to actual physical devices, data channel, and task assigned at all times.

The end of an I/O operation is usually signalled by an interrupt. However, some I/O managers on mini and microcomputers do not use interrupts – here, the I/O manager continually checks the status of the I/O operation, and does not return control to the calling program until the operation is complete.

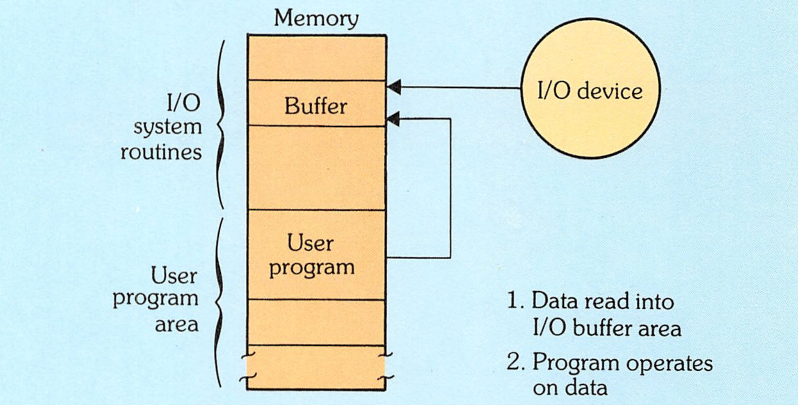
When performing I/O operations, the I/O manager reads data into a **buffer** which is an area of storage where data is held temporarily to facilitate transfer between devices operating at different speeds or on different time cycles. There is a trade off between buffer size and the speed of I/O operations: increasing buffer size increases the memory requirement and therefore cost, but may reduce the total operation time. *Figure 13* illustrates an I/O operation using a buffer.

Double buffering is used when data is input from one device and output to a second as illustrated in *figure 14*. Data is read from an output device into buffer A by an I/O processor until buffer A is filled. A second I/O processor writes this data from buffer A out to the output device. The first I/O processor then begins reading data into buffer B. In this way, input and output operations can be carried out almost simultaneously. Buffer A is being emptied while buffer B is being filled and when both operations are completed, the two buffers are exchanged; buffer B is then emptied while buffer A is being filled.

When the I/O resources are used by two or more processors in a multiprocessor system, more than one of them may be



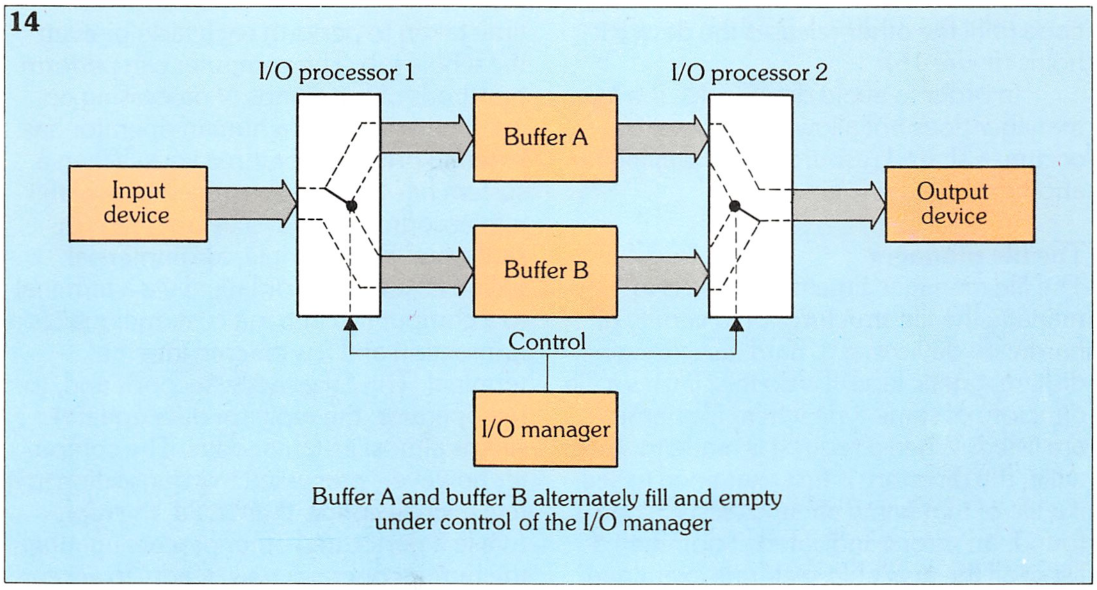
13



Above: a remote cashpoint terminal. (Photo: IBM).

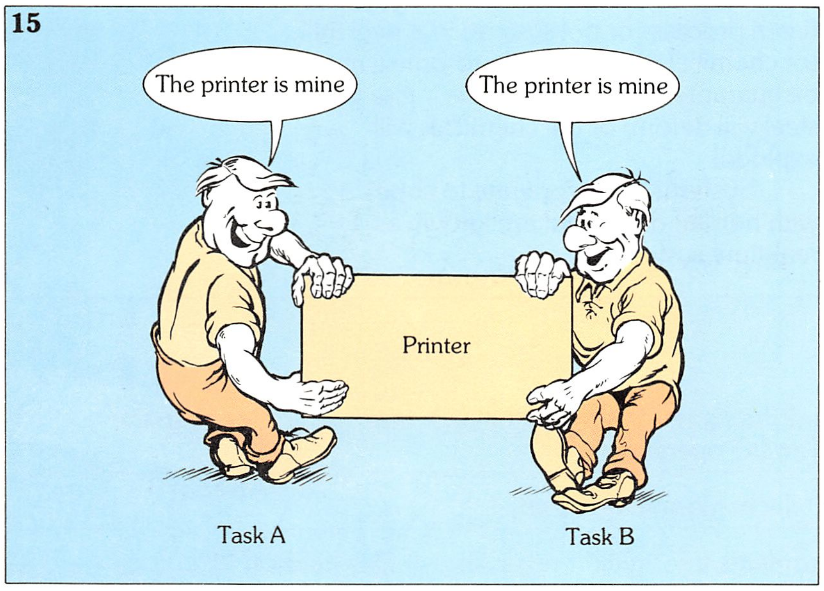
13. Input/output using a buffer.

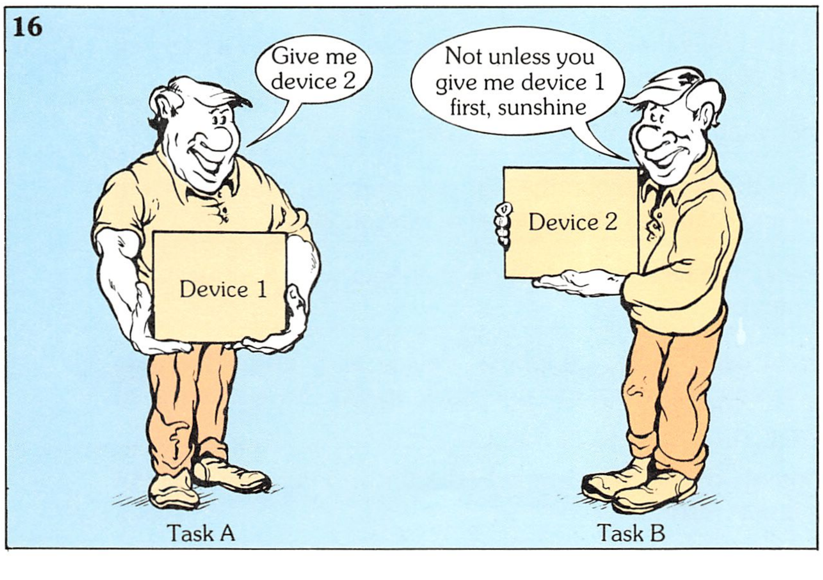
14. **Double buffering.**



15. **A race condition.**

16. **System dead-lock.**





assigned to the same I/O resource at the same time, unless precautions are taken. For example, suppose that the printer can be shared by tasks A and B, and that a flag, F, indicates whether the printer is free or not. Task A examines F, sees that it is zero and so sets F to 1; however, before A can change F, B also sees that F is zero and so sets F to 1. Both processes attempt to use the printer – the result is known as a **race condition** (*figure 15*).

To avoid this, most computers utilise a special hardware instruction which performs a **test and set** function. This instruction tests the flag, and if it is zero, sets it to one. Other processors are inhibited or prevented from referencing the memory location containing the flag until the test and set instruction is complete.

Another condition which could affect a system where different tasks were running at the same time is **system dead-lock**. A dead-lock occurs when two or more tasks need the same two or more resources, and can never get these resources. Suppose that task A needs devices 1 and 2. Task A requests device 1 and the I/O manager assigns this to device A; meanwhile process B requests and is assigned device 2. Task A now asks for device 2 – as this device is already assigned to process B, A retains device 1 and waits. B, which is still using device 2, asks for 1, but does not get it because it is assigned to A; task B then holds device 2 while waiting for device 1. Now, neither task can continue until the other releases the device it holds (*figure 16*).

In order to avoid dead-locks, the I/O manager does not allow a processor to occupy a shared resource while waiting for another shared resource.

The file manager

The file manager's main function is to manage the file structures on a variety of hardware devices, e.g. hard disk, floppy disk, magnetic tape. To do this, it uses a file directory of some type where file names are listed. When a request is made to store a file, the directory is first examined to see if a file of that name already exists: if one is found, an error is indicated, if not, then a list of all the available sectors is examined, and one or more are assigned to the file. The I/O manager is then instructed to store the data for the file into the assigned disk location, and the directory is updated to include this new file.

Timing in the computer system

When an operating system is running a program, the user has no 'sense' of the time taken to perform each task, or even the whole job. The computer can perform hundreds of thousands of processing operations in the time a human operator has taken to press two or three keys. When a system has been programmed to take this into account it is said to be operating in **real time**. For example, a commercial airline ticket agency is linked via a terminal to a computing centre, a customer makes a transaction and it is entered into the terminal. This takes a few seconds and, to the operator, the reply (or data update) seems almost instantaneous. The computer however, processed this transaction in a few microseconds, then 'held' the reply (while it performed other processing) until the human operator was 'ready' to accept it. Another example is a monitoring system (i.e. a process control system) in a steel mill (or chemical plant) where a response must be guaranteed in a certain time – else the steel will deform or the chemicals will explode!

Systems which operate to cope with human data input are known as real-time systems.

Glossary

channel status word	a block of information set up by the I/O manager describing the conditions of an I/O operation
data channel	a path along which information flows
file directory	a list which contains information regarding all files held on disk, their location and size
flag	an indicator that shows the existence of a certain condition when set; and the absence of that condition when clear
global variable	a variable which may be altered by more than one process
resource manager	a module within the set of programs comprising the operating system which controls and manages a specific set of operations
service request list	list or table associated with each resource manager where requests for servicing are queued
spooling	the operation of using a fast peripheral device as a buffer for data transferred between main storage and slow speed devices, such as printers
state vector	where the contents of the CPU are stored after an interrupt, for later use when the task is resumed

ELECTRICAL TECHNOLOGY

Trigonometry and sinusoids

Sinusoidal voltages are based on the two trigonometrical functions of an angle, the **sine** and the **cosine**. *Figure 1* enables us to define these functions. If we take the angle between the straight lines OP and OX to be θ and draw a perpendicular, PM, from P onto OX, the sine of the angle θ is given by:

$$\sin\theta = \frac{PM}{OP}$$

the cosine of θ is given by:

$$\cos\theta = \frac{OM}{OP}$$

A related geometric function, the **tangent**, can also be found where:

$$\tan\theta = \frac{PM}{OM}$$

As we can see from these equations, these trigonometric functions of θ are independent of the absolute length of the lines that make up the angle, and are derived from the *ratio* of these lengths. Looking again at *figure 1*, the line OQ is twice the length of OP; similarly, QN is twice the length of PM; and ON is twice the length of OM. However, the sine, cosine and tangent of angle θ remain the same.

For convenience, we shall therefore always think of, and draw, OP as one unit long. Then:

$$\sin\theta = PM$$

and:

$$\cos\theta = OM$$

If we now draw the line OY perpendicular to OX, and draw PL perpendicular to OY, we see that OL = PM, so:

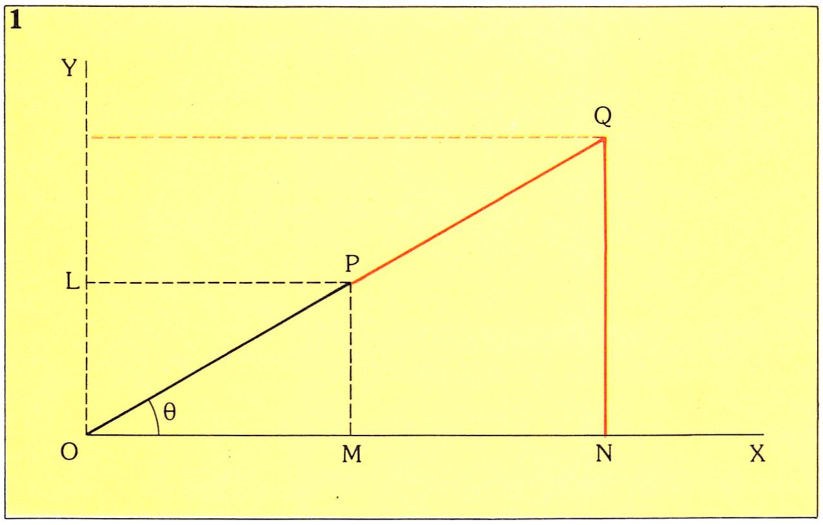
$$\sin\theta = OL$$

Thus we can think of the sine of an angle as being equal to the length of the shadow of a unit line at angle θ (OP) on the vertical line OY when illuminated by a light a long way out to the right. The line OL is therefore known as the projection of OP on OY. Similarly, the cosine of θ is the projection of OP on OX.

Using this idea we can see that when the angle θ increases from zero to 90°, the projection OL increases to its maximum value of 1; as θ goes from 90° to 180°, OL again falls to zero; from 180° to 270°, OL increases negatively to its maximum value of -1; and from 270° to 360°, OL again becomes zero. (Note that the distances measured upwards are positive and downwards are negative.)

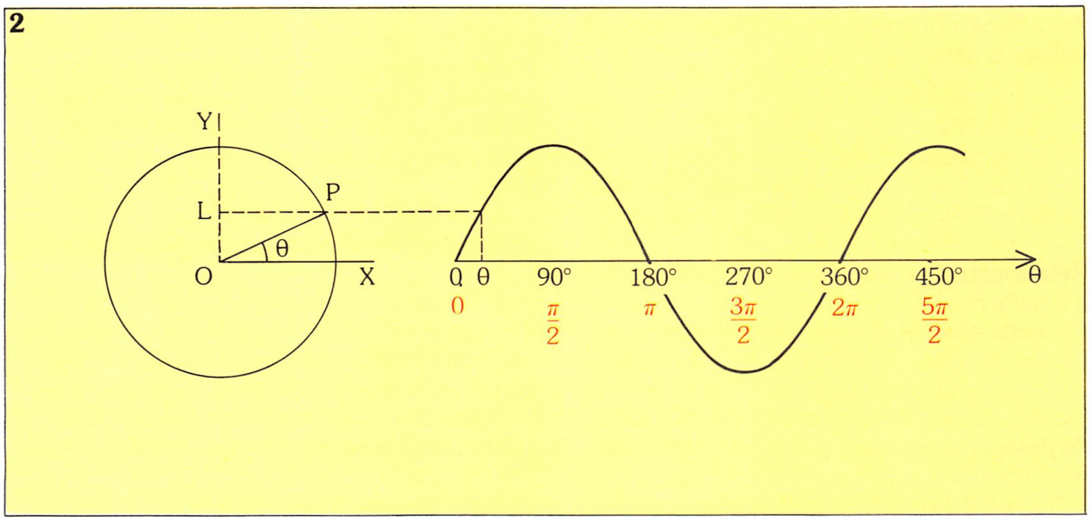
In *figure 2* we have used this method to plot the value of OL that corresponds to each particular value of θ. As θ increases, the value of OL changes – enabling us to trace out the curve shown on the right. This curve is

1



1. Defining the sine, cosine and tangent of angle θ.

2



2. Plotting the value of OL corresponding to each value of θ, produces a sinusoid.

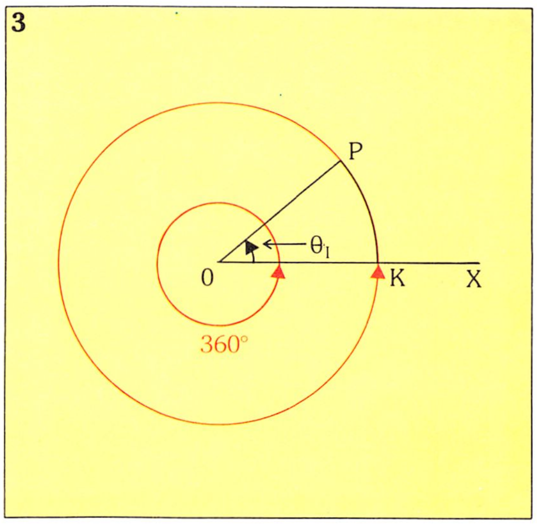
called a sinusoid. We can see that after 360° the curve repeats itself, as the angle returns to its starting point of zero degrees, so sin 450° = sin (360° + 90°) which is the same as sin 90°. In general terms we can say that:

$\sin(360 + \theta) = \sin\theta$

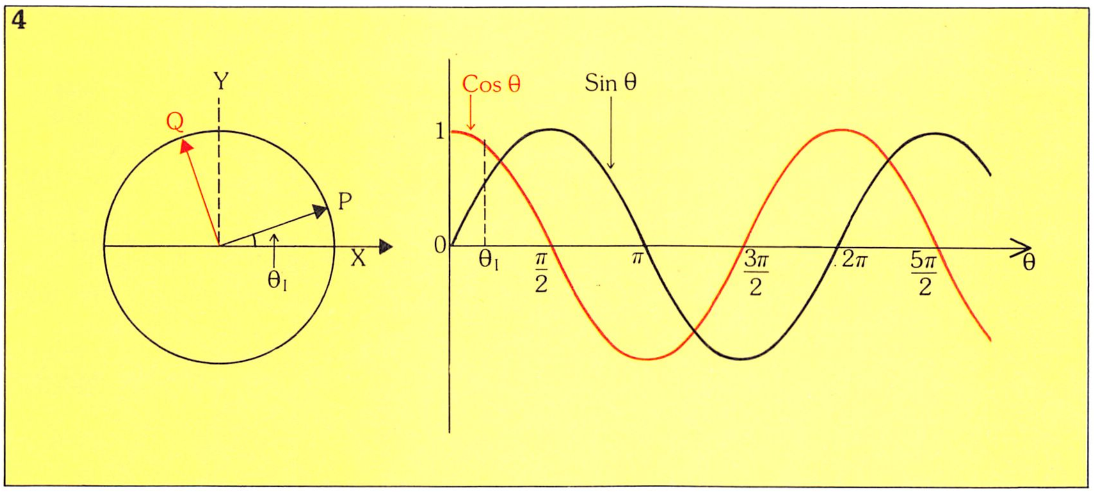
Degrees and radians

Most of us are accustomed to measuring angles in degrees, however, when alternating voltages are dealt with it is much more convenient to use the unit of **radians**.

In *figure 3*, the lines OP and OK delineating the angle θ_1 are both one unit in length. If we imagine a circle of 1 unit radius, whose centre is at 0, then the number of radians in the angle θ_1 is given by the length of the arc, KP.



3. The number of radians in the angle θ is given by the length of the arc, KP.



4. A sine and a cosine wave for angle θ.

We can compare the value of an angle in radians with that in degrees if we consider the angle 360° – which corresponds to a complete revolution. The arc of a circle that bounds this angle is the full circumference of the circle. As the circle has a radius of 1 unit, then the circumference has a length of 2π. So, 360° = 2π radians. For any angle $\theta°$, the value in radians is given by:

$$\theta^r = \frac{2\pi\theta°}{360°}$$

We have marked the angles measured in radians on *figure 2* in red, and from now on angles will always be measured in radians.

Relationship between sines and cosines

It is fairly obvious that a cosine curve will have the same general shape as a sine curve. However, when $\theta = 0°$, $\cos\theta = 1$, so the curve starts from the maximum value. As θ increases to $\pi/2$ the cosine (= OM) falls to zero. As distances measured to the right are considered positive and those to the left negative for values of θ from $\pi/2$ to π, the cosine becomes negative, rising to the maximum value of -1. *Figure 4* shows a sine and a cosine curve. We can see that not only does:

$\sin(2\pi + \theta) = \sin\theta$

but also:

$\cos(2\pi + \theta) = \cos\theta$

Further, we saw that the sine curve was traced out by the projection of the line OP on OY, which rotates with the increasing angle θ. Similarly, if we draw a line OQ perpendicular (at an angle of $\pi/2$) to OP, then the projection of this line will trace out the red cosine curve. Thus:

$\sin(\pi/2 + \theta) = \cos\theta$

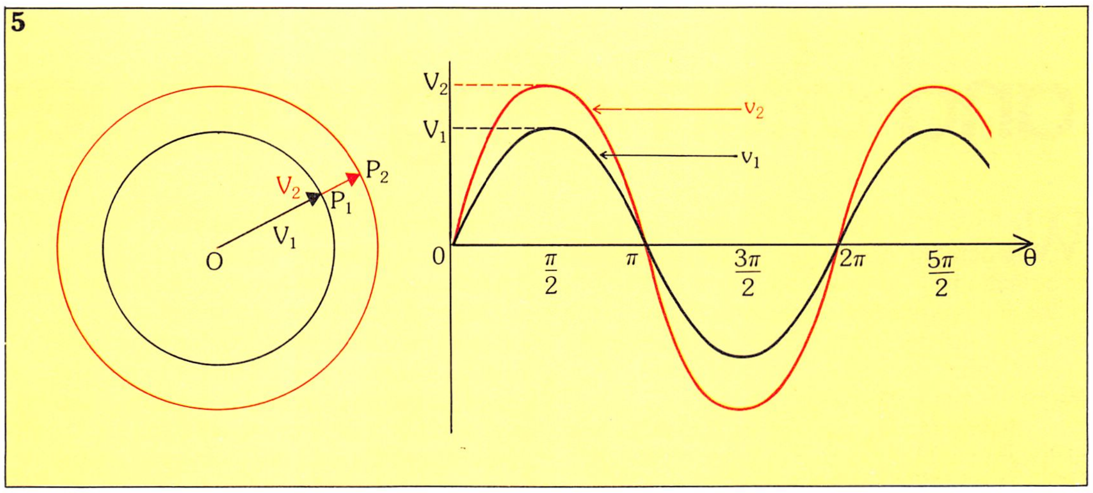
Both the sine and cosine curves are sinusoids.

Amplitude of a sinusoid

The amplitude of a sinusoid is given by the magnitude of the maximum value of the sine curve, measured positively or negatively from 0. *Figure 5* shows that the amplitude of the (black) sinusoid is given by the length V_1 of the line OP_1, whose projection on OY gives the shape of the sine wave. If the length V_2 is

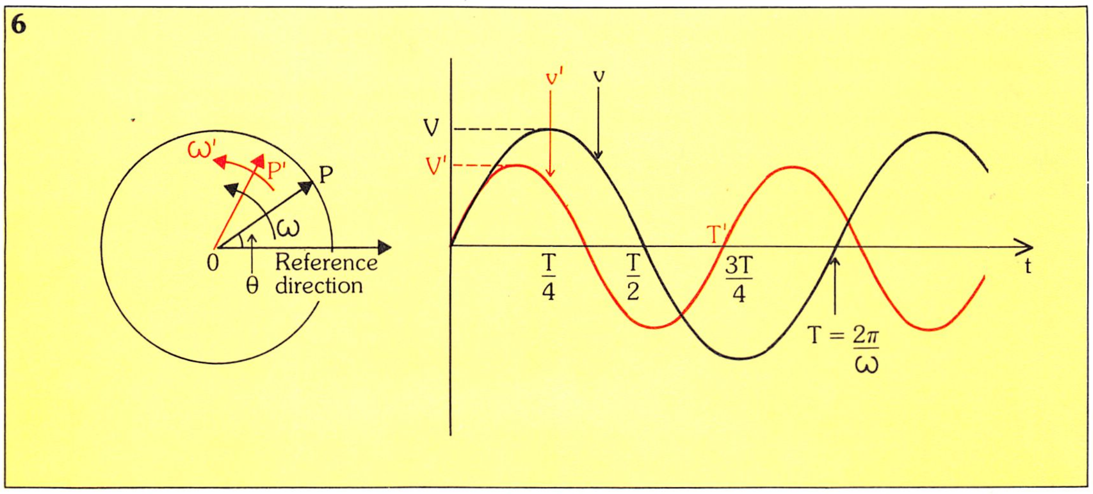
5. The amplitude of the sinusoid is given by the length V_1 of OP_1, whose projection on OY gives the shape of the sine wave.

5



6. Determining the frequency of a sinusoid.

6



increased to OP_2, we can see that the shape of the sinusoid remains the same, but it has a larger amplitude. If we use V_1 to give the value of the sine wave for any angle then we have:

$$V_1 = V_1 \sin \theta .$$

Frequency of a sinusoid

If we assume that the line OP, in *figure 6*, is rotating in an anticlockwise direction with a speed of rotation of f revolutions per second, then it will have turned through one complete revolution in a time T, where T = 1/f. (Note that we always consider anticlockwise to be the positive direction of rotation.)

Since one revolution is equal to 2π radians, we can say that OP is rotating at a speed of $2\pi f$ radians per second. We give this the special symbol ω (Greek lower case omega) where:

$$\omega = 2\pi f$$

If OP is lying along OX (the reference direction) when we start our measurements, then after time t it will be lying along a direction making an angle of θ radians with OX, where:

$$\theta = \omega t$$

We can therefore alter the scale on the horizontal axis so that it measures time, as in *figure 6*, where:

$$v = V \sin \omega t$$
$$= V \sin 2\pi f t$$

f is termed the **frequency** and is measured in Hertz (Hz); ω is the **angular frequency**, measured in radians per second ($rads^{-1}$); and T is the **period** of the wave in seconds.

If we consider another line OP′ (of length V′) rotating at a different angular frequency ω', then it will generate a sinusoid whose maximum value is V′ and has a frequency f′. Angular frequency ω' will of course be equal to $2\pi f'$, and the resulting sinusoid will take the form shown by the red curve in *figure 6*. The period of this sinusoid will be T′ = 1/f′. So, we can see that sinusoids of different frequencies will be generated by lines rotating at different speeds. □

Phasors and alternating voltages

We have now seen that a sinusoid can be generated by considering the projection of a rotating line onto a vertical axis. The length of the line is equal to the maximum value of the sinusoid, and the speed of rotation (measured in revolutions per second) is equal to the frequency of the sinusoid.

In a previous *Basic Theory Refresher*, we also found out that a coil rotating in a uniform magnetic field will generate a sinusoidally shaped EMF. Thus, we can see how sinusoids derived from trigonometry are related to voltages and currents in an electric circuit.

We can use the rotating line principle to give us a simple representation of a sinusoidal voltage. A sinusoidal voltage v of frequency f and maximum value $\hat{V}$ is represented by:

$v = \hat{V} \sin 2\pi ft$

which corresponds to a line of length $\hat{V}$, that rotates about one end with an angular velocity of f revolutions per second (or $2\pi f$ radians). These lines are called **phasors**, and the relationships between different **phasors** will show us how we can add and subtract different sinusoidal voltages and currents.

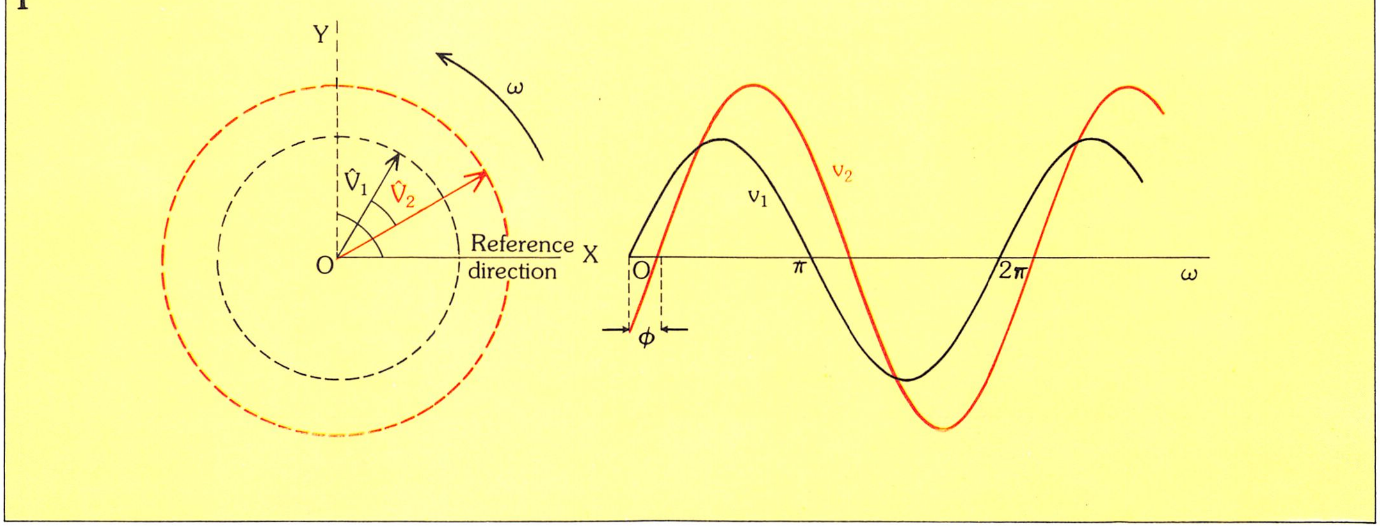
As practically all the circuits that we shall be studying use voltages and currents of the same frequency, we will not need to state the speed of rotation for each phasor. We only need to know the frequency at the start of each case considered, and we can then assume that it remains at this value throughout.

If you imagine a photograph of the phasors taken at any two instants in time then we know that the overall position of the phasors may have changed, but their position relative to each other will be the same. The length of the phasors and their relationship to one another is sufficient to tell us all we need to know about the alternating sinusoidal currents and voltages in a circuit. We must now consider how two different sinusoidal voltages are represented by phasors.

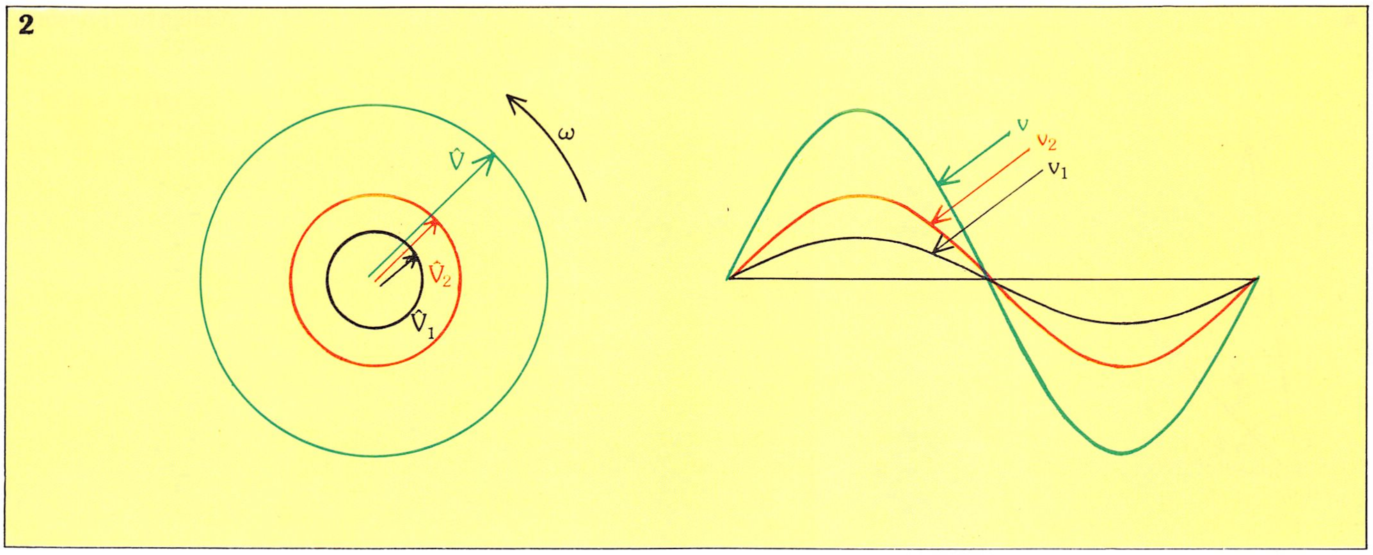
Phase relationship of sinusoids
Figure 1 shows two phasors $\hat{V}_1$ and $\hat{V}_2$ rotating at a speed of ω radians per second. Phasor $\hat{V}_2$ lags behind phasor $\hat{V}_1$ by an angle of ϕ. We shall take our time origin (zero point) to be at the point when phasor $\hat{V}_1$ lies along the reference direction. As the speed of rotation, and hence the frequency, is constant, we can multiply all units on our time axis by ω and plot the sinusoid voltage v_1 on the axis of ωt. Each cycle occurs over an interval of 2π. Now, when ωt is zero, the value of voltage v_1 is 0, but the value of voltage v_2 (the projection of the phasor $\hat{V}_2$ on OY) is negative – as shown by the red curve. Phasor $\hat{V}_2$ will only lie along the reference direction when $\omega t = \phi$. Voltage v_2 will then be 0 at this point.

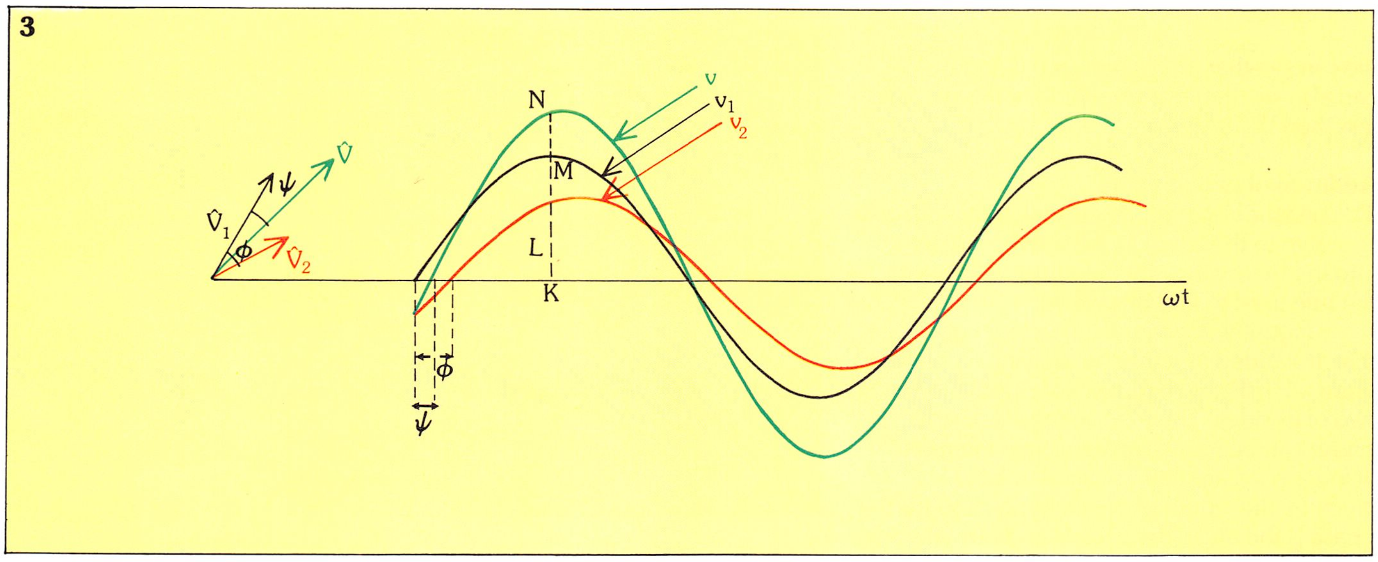
We can see that phasor $\hat{V}_1$ will draw a

1



1. Two phasors rotating at a speed of ω radians per second.





2. Two in phase sinusoid curves, v_1 and v_2 and their sum, v.

3. Sinsusoids of *figure 2* shown together with their phasors.

sine wave of amplitude $\hat{V}_1$, and that phasor $\hat{V}_2$ will draw a sine wave of amplitude $\hat{V}_2$, displaced from sine wave $\hat{V}_1$ by an angle ϕ on the ωt axis. Voltage v_2 rises to its maximum later in time than voltage v_1 and so we can say that voltage v_2 **lags** voltage v_1 by a **phase angle** ϕ. Alternatively we could say that voltage v_1 **leads** voltage v_2 by the phase angle ϕ. We can write these voltages as follows:

$$v_1 = \hat{V}_1 \sin \omega t$$
$$v_2 = \hat{V}_2 \sin (\omega t - \phi)$$

Adding sinusoids or phasors

Figure 2 shows two sinusoidal curves, v_1 and v_2 of the same frequency and with amplitudes $\hat{V}_1$ and $\hat{V}_2$. These two sinusoids are in phase – i.e. the phase angle between them is zero. The sum, v, of the two sinusoids is given by adding their values together at every point in time, thus, $v = v_1 + v_2$. We can immediately see that the curve of v is a sinusoid and its amplitude, $\hat{V}$, is given by:

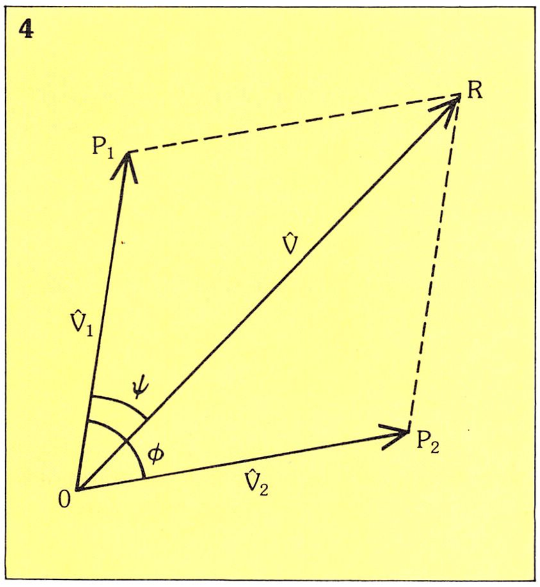
$$\hat{V} = \hat{V}_1 + \hat{V}_2$$

Now let's consider two voltages v_1 and v_2, where v_2 lags v_1 by an angle ϕ. Thus:

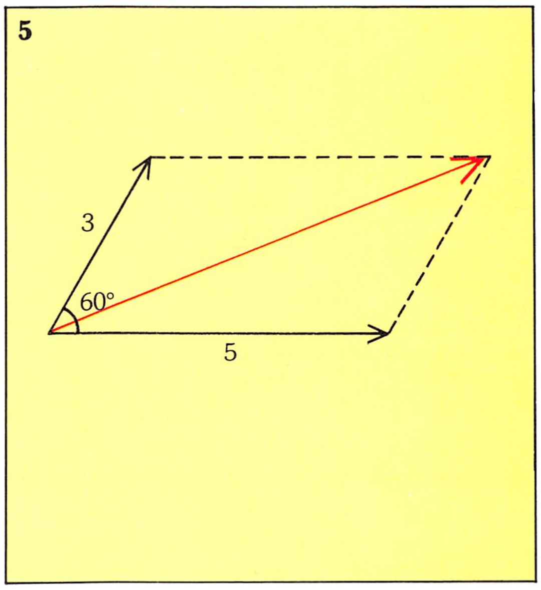
$$v_1 = \hat{V}_1 \sin \omega t$$
$$v_2 = \hat{V}_2 \sin (\omega t - \phi)$$

These sinusoids and their phasors are drawn in black and red in *figure 3*. We can add these two sine waves together by taking some value of ωt, say at point K, and adding the value of sinusoid v_1 (=KM) to the value of sinusoid v_2 (=KL), to give the total voltage $v = v_1 + v_2$ (with KN = KM + KL). If we do this at every point along the ωt axis we shall be able to draw the voltage v (shown in green) which is the sum of voltages v_1 and v_2. We can see that this is a sinusoid of the same frequency as the original voltages, and that it passes through zero at a time later than that of sinusoid v_1 and earlier than that of sinusoid v_2. If we call its phase angle Ψ, then

4



5



4. Adding two phasors, $\hat{V}_1$ and $\hat{V}_2$.

5. Finding the sum of two phasors, $\hat{V}_1$ of magnitude 5, and $\hat{V}_2$ of magnitude 3.

we can see that Ψ lies between 0 and ϕ. We can also see that the amplitude of this wave is less than $\hat{V}_1 + \hat{V}_2$.

Addition of two phasors

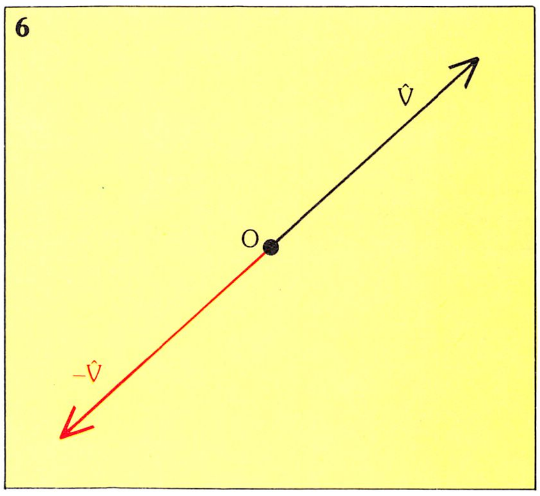
The phasor $\hat{V}$ that represents the total voltage v is shown to the left of *figure 3*. Without going into the theory in great detail, we will look at the rule used to find this value.

Figure 4 shows two phasors $\hat{V}_1$ and $\hat{V}_2$, that are added together by drawing a parallelogram OP$_1$RP$_2$, with phasors $\hat{V}_1$ and $\hat{V}_2$ forming two of its sides. The total voltage $\hat{V}$, and its phase can be found by drawing in the parallelogram's diagonal. The total voltage $\hat{V}$ is given by the length of the diagonal, and the phase is the angle between $\hat{V}$ and our reference $\hat{V}_1$. We can now see that the total voltage lags behind $\hat{V}_1$ by the angle Ψ.

Let's look at an example. The reference phasor is usually drawn horizontally, but this is purely a matter of convenience. Remember, the phasor diagram is only a 'photograph' taken at an arbitrary moment, of the whole diagram rotating about point 0. Let's say that we want to find the sum of two phasors $\hat{V}_1$ of magnitude 5, and $\hat{V}_2$ of magnitude 3; $\hat{V}_2$ leads $\hat{V}_1$ by 60°. These are drawn accurately to scale in *figure 5*. Drawing in the diagonal and measuring its length tells us that the total voltage is of magnitude 7. The angle Ψ, between the diagonal and the reference direction is 21.8°. Thus the sum of these two phasors is a phasor of magnitude 7, leading phasor $\hat{V}_1$ by 21.8°.

Three or more phasors can be added together in the same way, by taking them two at a time, then adding this sum to the next, and so on.

6



6. The negative of a phasor $\hat{V}$ is given by $-\hat{V}$.

Negative phasors

Figure 6 shows that the negative of a phasor $\hat{V}$, is given by $-\hat{V}$, which is drawn to the same length as $\hat{V}$, but in the opposite direction. The difference of two phasors, namely:

$$\hat{V} = \hat{V}_1 - \hat{V}_2$$

can be found by writing this equation as an addition:

$$\hat{V} = \hat{V}_1 + (-\hat{V}_2)$$

so to subtract phasors, we negate one phasor and then add them together. □

19
SOLID STATE ELECTRONICS

More multilayer semiconductor devices

Diacs

The multilayer semiconductor structure discussed in *Solid State Electronics 18* is used to make thyristors and triacs. These are often used in applications where power is to be switched or regulated upon application of trigger pulses. It is to the subject of the generation of these trigger pulses that we now turn.

The electronic devices used to produce thyristor and triac trigger pulses are often multilayer devices, too, and have very similar structures to thyristors and triacs. The first such device is the **diac** (diode alternating current). Its structure is shown in *figure 1a* and we can see that it is a five-layer, two-terminal device. If we imagine the device divided into two halves (left and right) then we may consider it as two electrically separate, but physically connected, four-layer diodes in parallel and reversed. The circuit symbol for a diac (*figure 1b*) shows this.

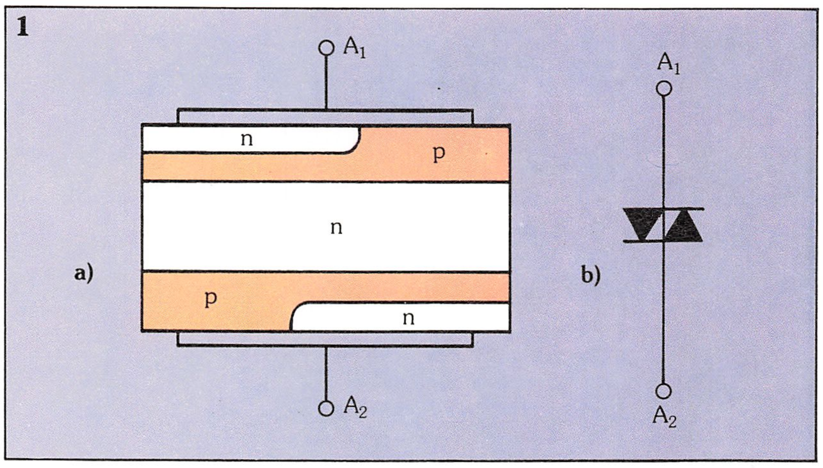
Because of its parallel structure, a diac is a bidirectional device. In fact, its characteristic curve, shown in *figure 2*, is symmetrical about the I-V axes. If the voltage applied across it is below either direction's **breakover voltage** (this is, in fact, the Zener breakdown voltage of the diode, but it is known as the breakover voltage in diacs), $+V_{BO}$ or $-V_{BO}$, the device is off. However, if the applied voltage in either direction is greater than either breakover voltage, the device turns on – the current through it rapidly increases, while the voltage across it decreases. Once on, the diac can only be turned off by removing the applied voltage, even if only for an instant.

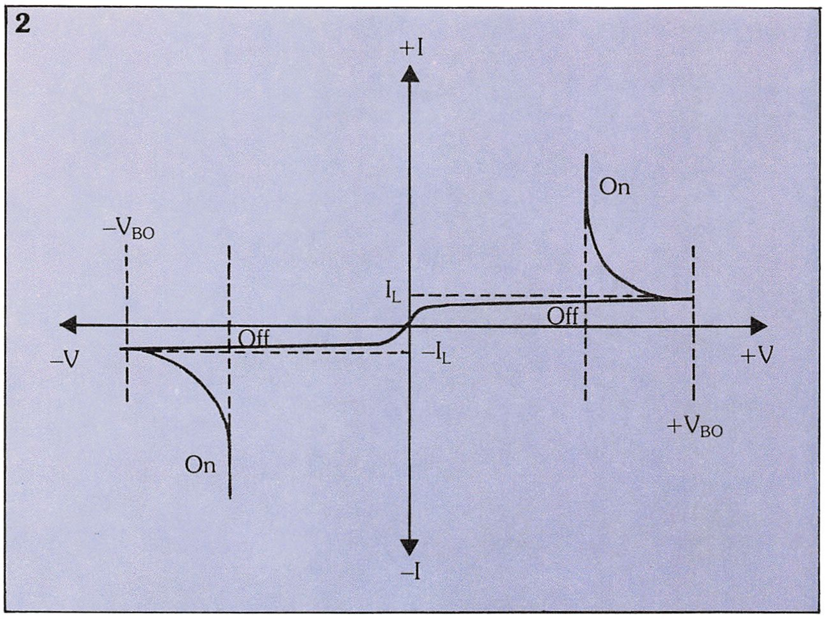
A good example of the use of a diac to generate trigger pulses is in a power supply. *Figure 3a* shows a simple type of power supply, based on those often found in television receivers. In it, a thyristor is used to switch and regulate the voltage applied to the load (in this example, part of the television receiver's circuitry). Gate trigger pulses to the thyristor are generated by the circuit formed by resistors R_1 and R_3, variable resistor R_2, capacitors C_1 and C_2 and the diac.

We can see how the power supply works if we first assume that the applied 240 V AC mains input is at 0 V (t_0 in *figure 3b*). The voltages at points X and Y in the

1. (a) Structure of a diac; (b) its circuit symbol.

2. Characteristic curve of a diac.

1


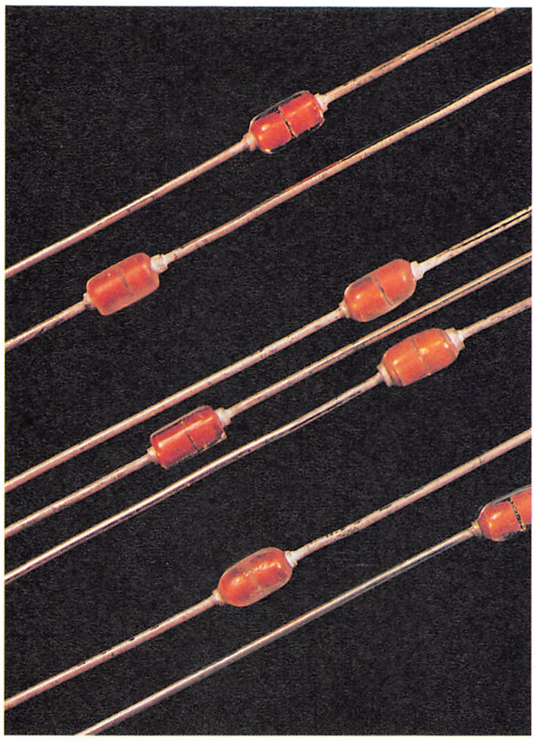
2


circuit can be assumed to be zero. As the applied mains voltage increases (from t_0 to t_1) in a positive half-cycle, the voltage at point X also increases because the capacitor C_1 is charged by current through resistors R_1 and R_2. Meanwhile, the voltage at point Y remains at zero.

When the voltage at point X increases so that the voltage across the diac reaches the breakover voltage V_{BO}, the diac turns on, rapidly increasing the voltage at point Y, thus providing a trigger pulse to the thyristor gate. This is seen at time t_t, in *figure 3c*. *Figure 3d* shows the regulated output voltage to the load.

As we saw in the last chapter, the point in time when the thyristor trigger pulse occurs defines the thyristor circuit's output voltage. Obviously, the output voltage must therefore be dependent on the time taken to turn on the diac after the mains positive half-cycle begins. This time is, in turn, dependent on the charging current of capacitor C_1, which can be adjusted by altering variable resistor R_2. The value of variable resistor R_2, therefore defines the output voltage.

The variable resistor used in such a TV power supply is generally a **preset resistor**, i.e. a resistor which is adjusted at the factory and not accessible by the user.


Neville Miles

Unijunction transistors

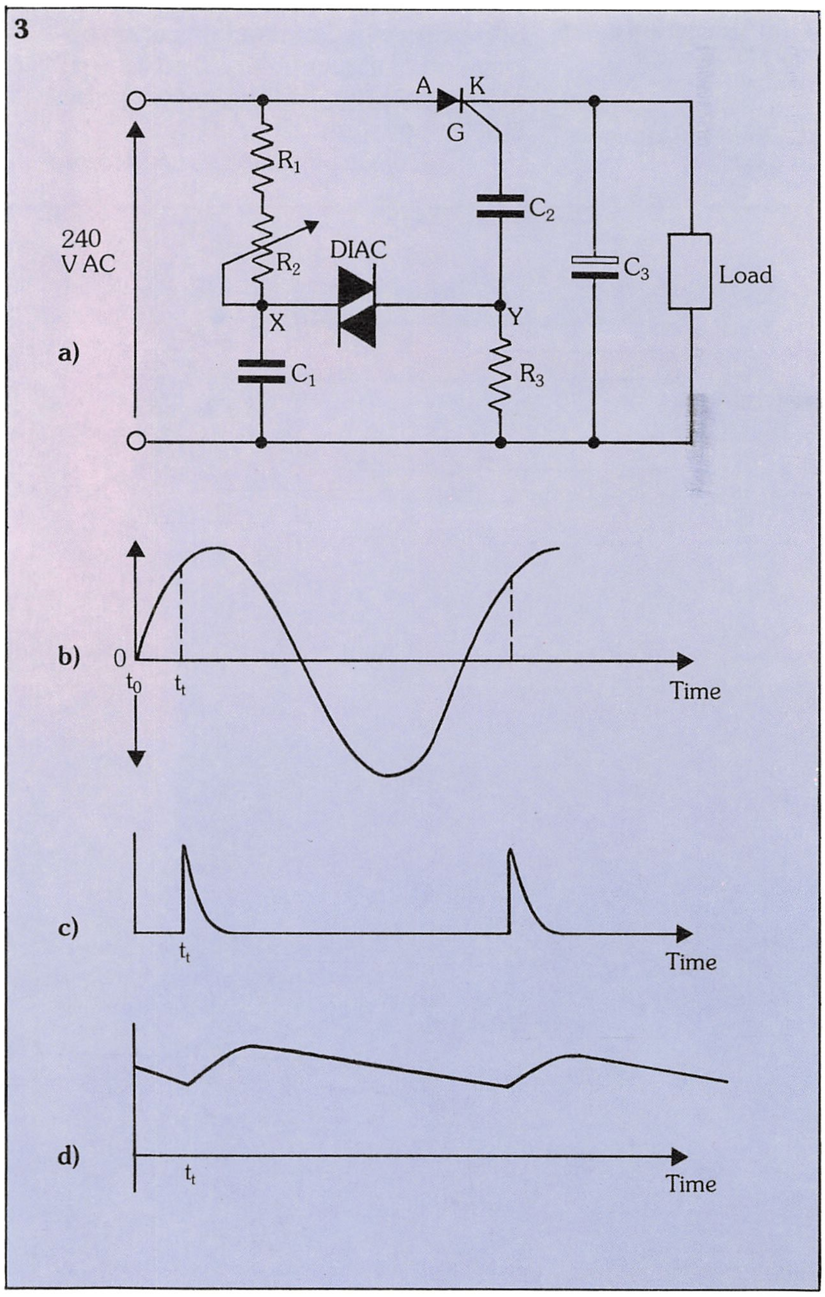
A **unijunction transistor** (UJT) is a three-terminal device, consisting of a bar of doped silicon, with an off-centre single p-n junction as is shown in *figure 4a*; its circuit symbol is shown in *figure 4b*. UJTs with a p-type bar can be made – their circuit symbol's arrowhead points in the opposite direction to the arrowhead for an n-type bar. Contacts are made to the n-type bar by terminals at both ends, called **base one**, B_1, and **base two**, B_2. The contact at the p-n junction is the **emitter**, E.

3. (a) A simple power supply circuit using a diac to trigger the thyristor; **(b)** the applied 240 V mains AC input; **(c)** the trigger pulses at time t_t; **(d)** the output voltage across the load.

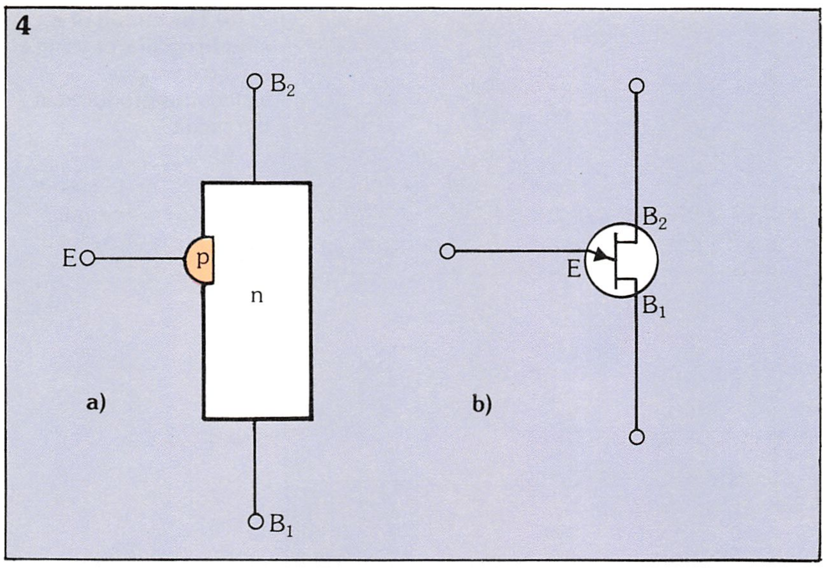
4. (a) Structure of a UJT; (b) its circuit diagram.

Below: diacs.

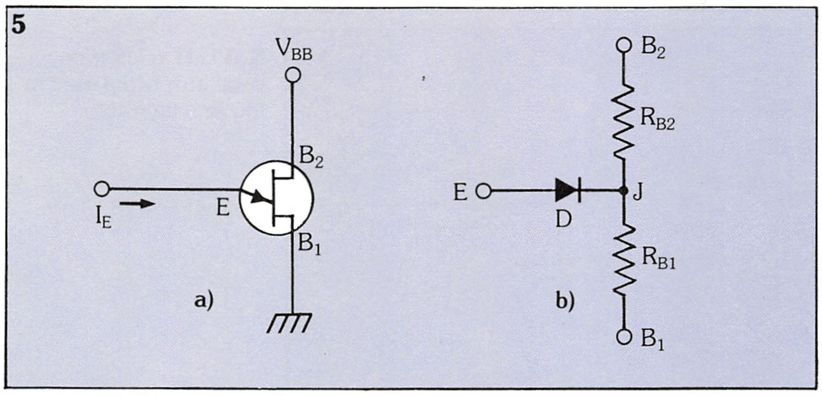
3



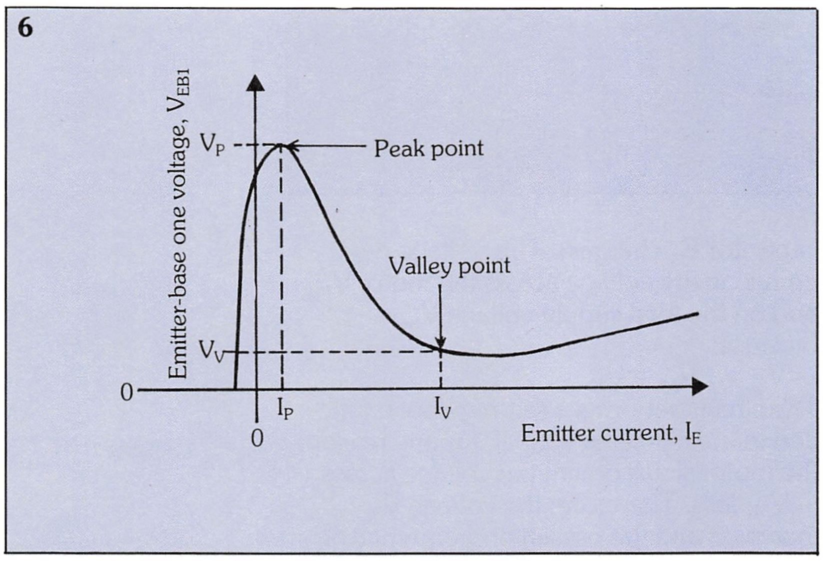
4



5



6



5. (a) A UJT in a basic circuit; (b) an equivalent circuit.

6. Emitter characteristic for a UJT.

Although the structure of a UJT is not multilayer (it is, in fact, close of that of a FET), it does operate in a similar way to multilayer devices, and is often used to produce trigger pulses in thyristor and triac circuits.

Characteristics

A UJT is shown in a basic circuit in *figure 5a* and an equivalent circuit is shown in *figure 5b*. If the emitter junction is reverse biased, the transistor acts as a simple resistor (of resistance $R_{B2} + R_{B1}$). The voltage at point J, the emitter junction, is given by the potential divider formula:

$$V_J = V_{BB} \frac{R_{B1}}{R_{B_1} + R_{B2}}$$

We can simplify this to:

$$V_J = \eta V_{BB}$$

where η is the **intrinsic standoff ratio**, which has a value between about 0.45 and 0.8, depending on the device.

However, if the emitter voltage rises above:

$$V_p = V_J + 0.7\,V$$

the emitter junction becomes forward biased and an emitter current flows from the emitter to base one. Holes are injected into the base one region, which flow down the semiconductor bar, causing a corresponding decrease in resistance of the base one resistor, R_{B1}. This, in turn, reduces the emitter voltage.

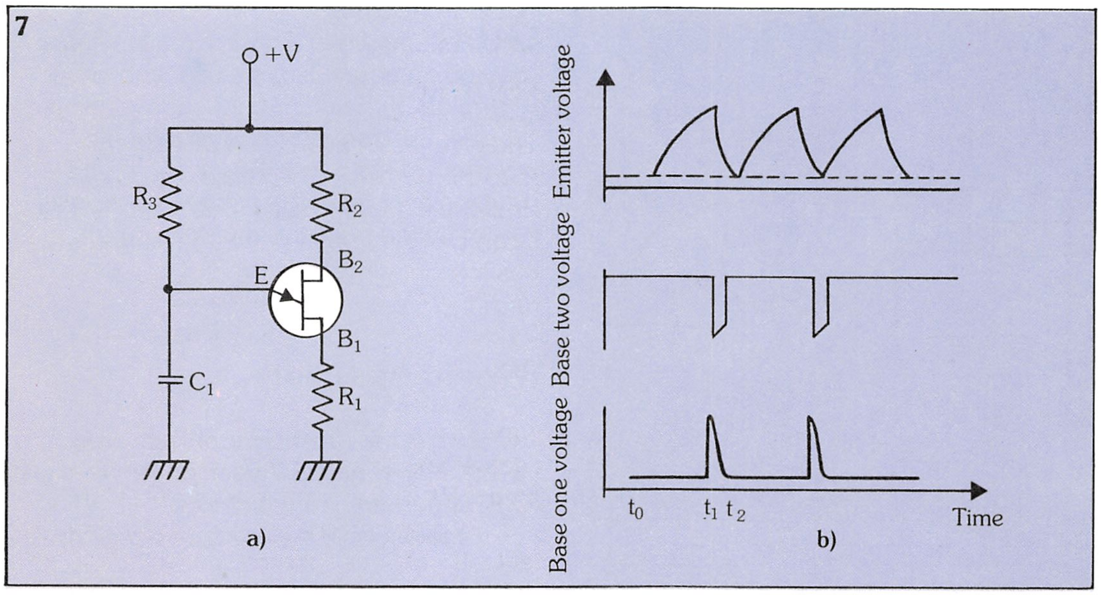
An emitter characteristic is produced as shown in *figure 6*. The **peak point** on the curve shows the **peak voltage,** V_p, at which the UJT switches from off to on. At current and voltage values between the peak point and the **valley point**, UJTs display a **negative resistance characteristic**, i.e. increases in emitter current produce decreases in the emitter-base one voltage. Above the valley point, a UJT is said to be saturated.

UJT applications

Unijunction transistors, because of their operating characteristics, are ideal generators of pulses, or **sawtooth waveforms**. The output voltages of such **oscillating circuits** are often used to produce thyristor and triac trigger pulses.

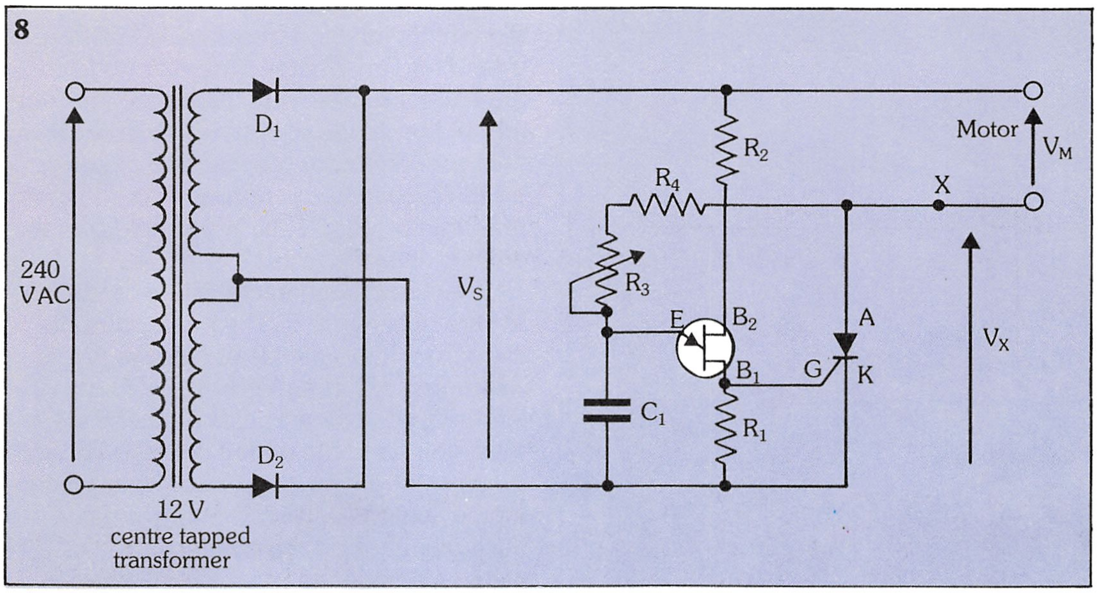
Figure 7a, for example, shows the circuit of a simple oscillator which uses a UJT. Initially (at time t_0) the emitter is reverse biased and so the UJT is off. Capacitor C_1 begins charging via resistor R_3, until (at time t_1) the voltage across it equals the peak voltage of the transistor, V_p. At this point, the transistor turns on and the emitter-base one resistance falls, dis-

7



7. (a) The circuit of a simple oscillator using a UJT; **(b)** various waveforms produced in the circuit.

8



8. A UJT relaxation oscillator being used to trigger a thyristor.

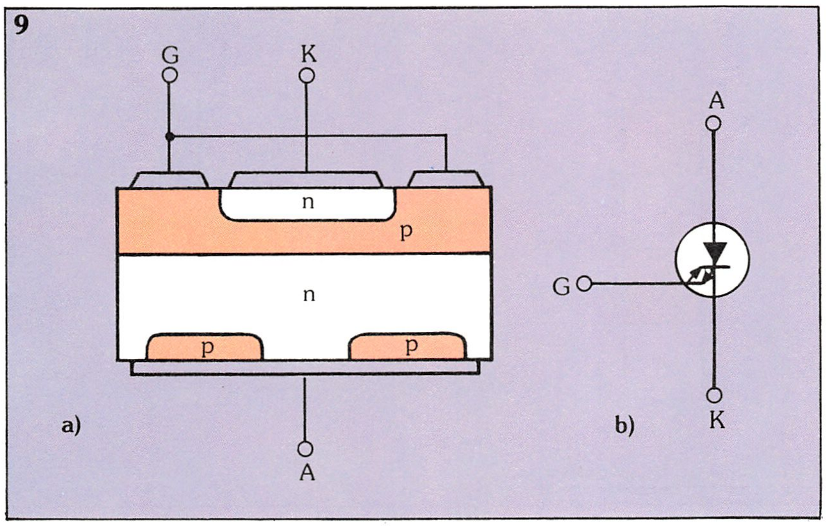
charging the capacitor. This turns the transistor off again (at time t_2), and the capacitor again begins to charge; the cycle is repeated indefinitely. Various waveforms produced in the circuit are shown in *figure 7b* and we can see how both pulses and sawtooth oscillations are produced. This type of oscillator is classified as a **relaxation oscillator**.

Figure 8 shows a similar UJT relaxation oscillator used to trigger a thyristor in a simple motor-speed control circuit for small DC permanent magnet motors – a model train motor, for example. The frequency of oscillation depends on the set value of variable resistor R_3, and on the voltage at point X (the higher the voltage, the faster capacitor C_1 charges). This voltage depends on the voltage across the motor V_M, and on the total supply voltage V_S, because:

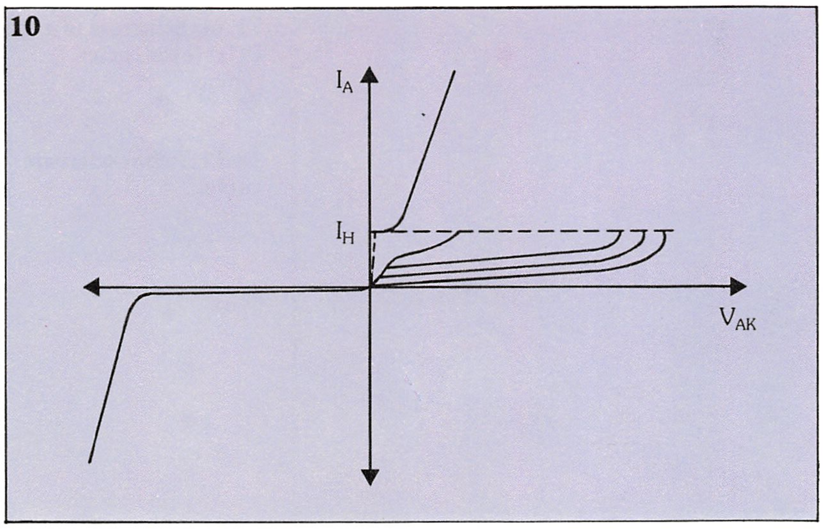
$$V_S = V_M + V_X$$

The circuit performs a self-regulating function in the following way. If, for any reason, the motor slows down, the voltage across it, V_M, falls. Therefore, the voltage V_X increases and the oscillation frequency of the UJT oscillator is increased. The thyristor is triggered earlier on in the positive half-cycle of supplied power, which increases the voltage across the motor, increasing the motor's speed. The motor may be set at the required constant speed by altering the value of variable resistor, R_3.

9



10



Gate turn-off semiconductors

The **gate turn-off thyristor** (GTO) is a four-layer device which operates in a similar way to an ordinary thyristor, in that it can be turned on by the application of a positive gate pulse, but it can also be turned off *by a negative gate pulse. Figure 9a* shows the GTO structure and *figure 9b* shows its circuit symbol.

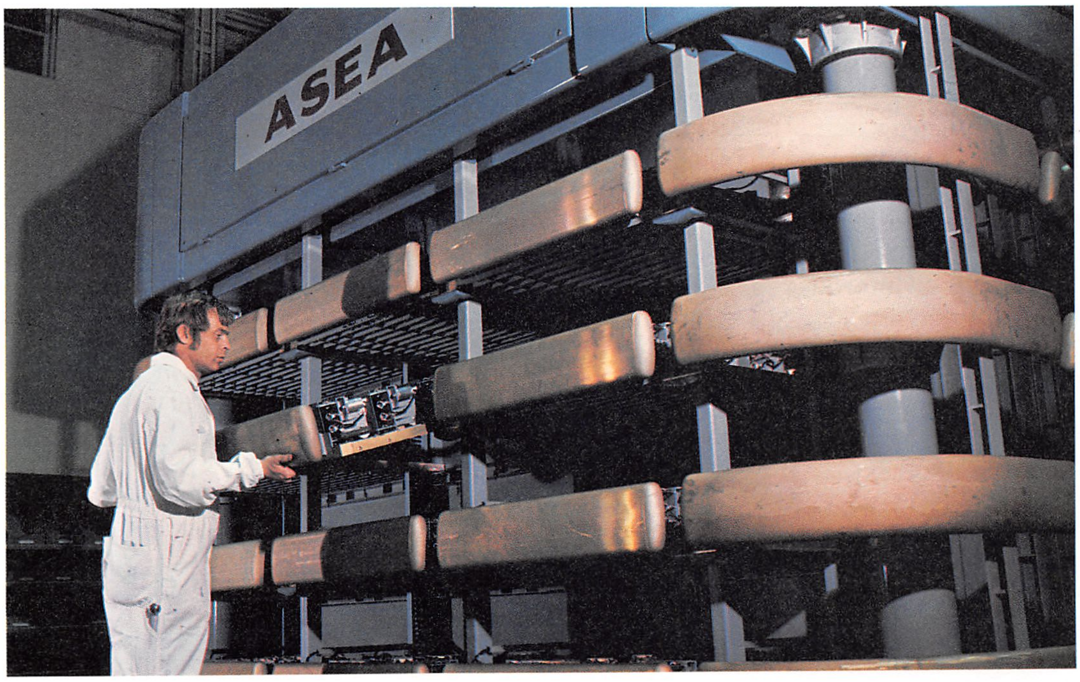
A collection of typical characteristic curves of anode-cathode voltage, V_{AK}, against anode current, I_A, is shown in *figure 10*. The advantages of a GTO are:
1) Low gate current: a gate pulse of only a few milliamps can switch several amps.
2) Short turn-on and turn-off times: gate pulses of only a few microseconds are required to turn the device on or off.
3) Low gate voltage: gate voltages of only a few volts can turn a GTO on and off.
4) Simplicity of use.
5) High voltages and current can be switched: GTOs can be manufactured which can operate with high voltages (up to about 2500 V) and high currents (about 1000 A).

GTOs can be used in all thyristor applications but the advantage of being able to turn the device off with a gate pulse, makes the GTO even more versatile.

9. (a) Structure of a GTO; (b) its circuit symbol.

10. A collection of typical GTO characteristic curves.

Right: part of an ASEA thyristor valve for the Skagerrack HVDC power transmission link between Norway and Denmark.



The Research House/ASEA

Programmable unijunction transistors

Although its name suggests otherwise, the **programmable unijunction transistor** (PUT) is a thyristor. It has a four-layer semiconductor structure as shown in *figure 11a*, differing only from the standard thyristor in that the gate terminal is connected to the n-type layer near the anode, rather than the p-type layer near the cathode. The circuit symbol of a PUT (*figure 11b*) shows this difference.

Together with two external resistors, a PUT can be programmed to generate an I-V characteristic curve similar to that of a basic UJT. All the unijunction parameters, e.g. η, R_{B1}, R_{B2}, V_p, may be varied by altering the two external resistor values. A PUT characteristic curve is shown in *figure 12* and the circuit of *figure 13* shows how the external resistor and the PUT are combined to form the equivalent of a UJT.

The total **interbase resistance** of the 'UJT' is approximately:

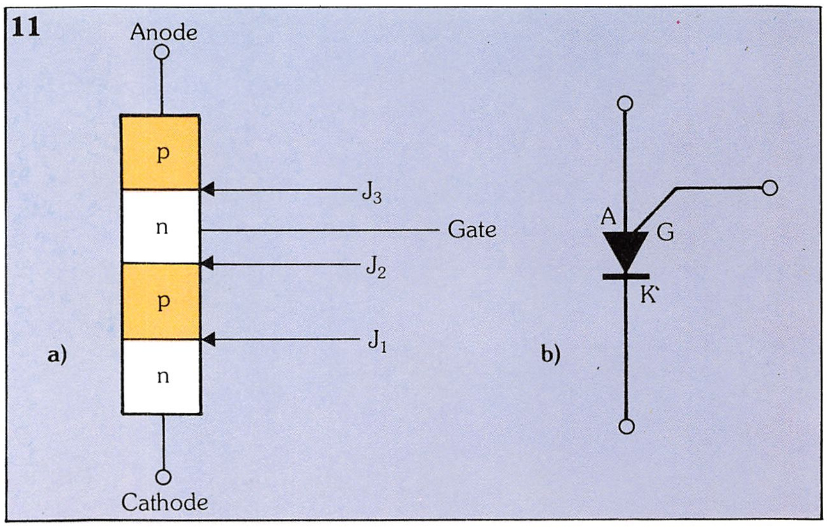
$$R_{B1B2} = R_1 + R_2$$

and η is approximately:

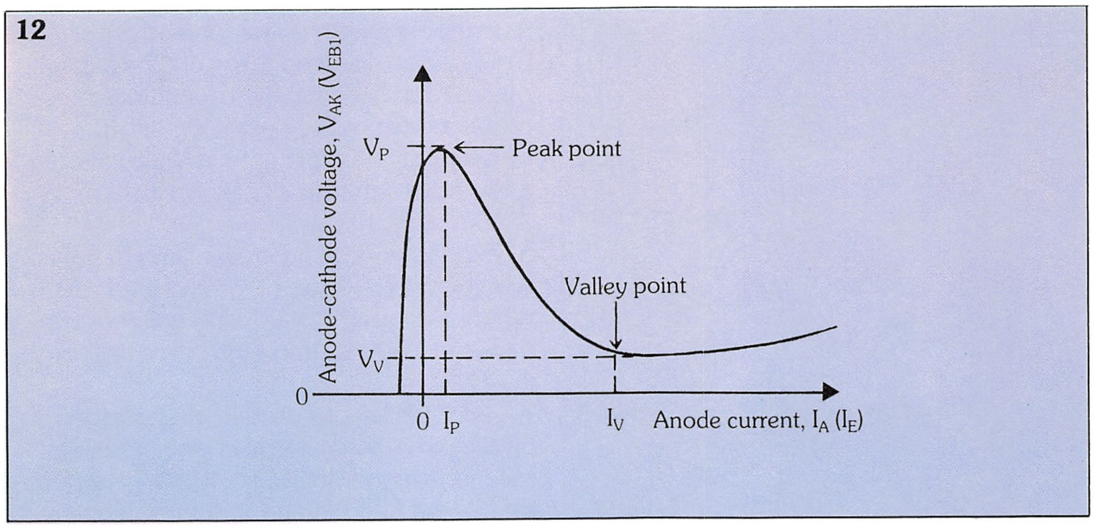
$$\eta = \frac{R_1}{R_1 + R_2}$$

A relaxation oscillator, similar in operation to the UJT oscillators of *figures 7* and *8*, is shown in *figure 14a*. *Figure 14b* shows waveforms at the indicated points in the circuit.

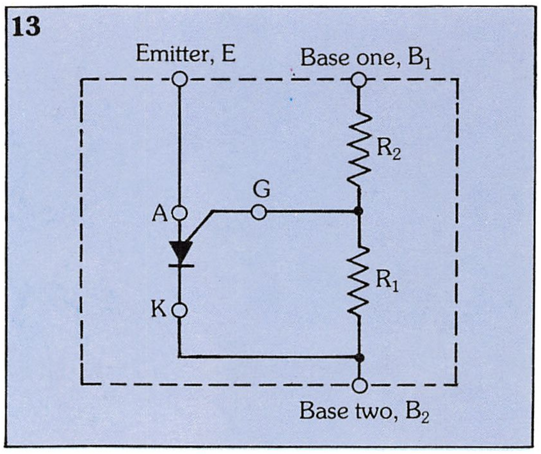
Capacitor C_1 charges initially through resistor R_3, from the applied supply voltage, + V. When the voltage at the PUT anode reaches the peak voltage, V_p, the anode-gate p-n junction, J_3, becomes forward biased and the PUT turns on. The capacitor C_1 now discharges through the

11


11. (a) Structure of a PUT; (b) its circuit symbol.

12


12. PUT characteristic curve.

13


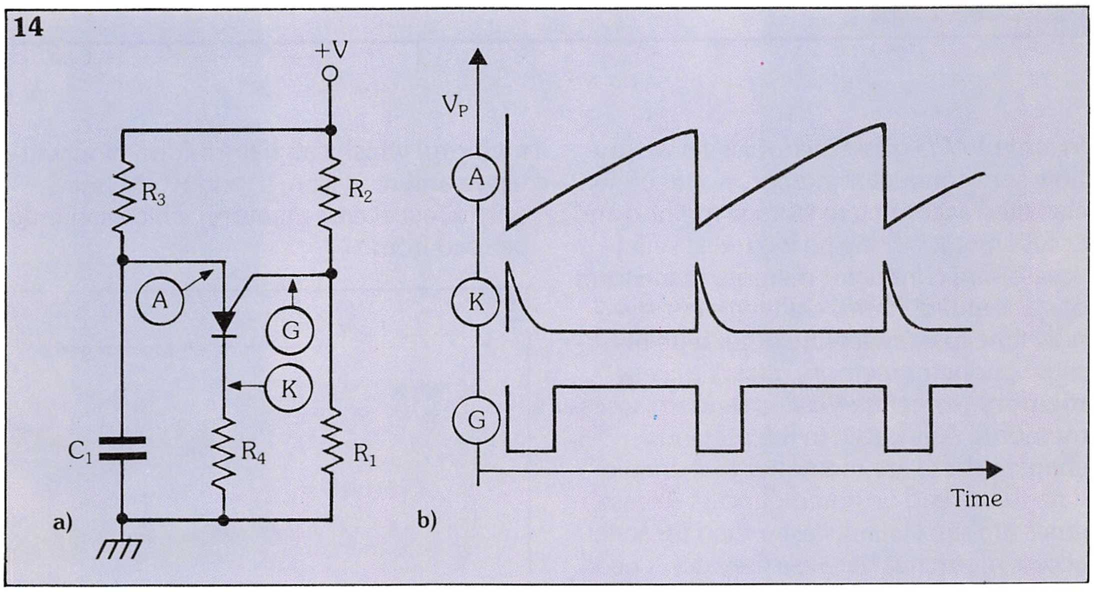
13. How an external resistor and a PUT are combined to form the equivalent of a UJT.

interbase resistance and resistor R_4, causing a positive-going and rapidly decaying pulse at the cathode, and a negative-going pulse at the gate. When the voltage across capacitor C_1, falls below the value:

$$V_{C1} = V_V + V_{R4}$$

the PUT switches off, and the cycle restarts.

14



14. (a) A relaxation oscillator; (b) waveforms at the indicated points.

Glossary

base one, base two, emitter	the three terminals of a unijunction transistor
diac	a five-layer, two-terminal semiconductor device which acts as two parallel bidirectional four-layer devices, turning on only when the breakover voltage in either direction ($+V_{BO}$ or $-V_{BO}$) is exceeded
intrinsic standoff ratio, η	the ratio $(V_p - 0.7)/V_{BB}$ of a unijunction transistor, where V_p is the peak voltage of the device and V_{BB} is the applied voltage between base one and base two terminals
gate turn-off (GTO)	type of thyristor which can be turned off by application of a negative gate pulse
negative resistance characteristic	a characteristic displayed by UJTs and multilayer semiconductor devices where an increase in current produces a decrease in voltage
peak point	the point on a UJT or PUT characteristic curve where the device turns on for an increase in current, producing a decrease in voltage
programmable unijunction	a thyristor which can be programmed with the values of two external resistors, to produce a characteristic curve similar to a UJT's characteristic curve
unijunction transistor (UJT)	a three-terminal semiconductor which has only one junction and has a negative resistance characteristic over part of its operating range

17

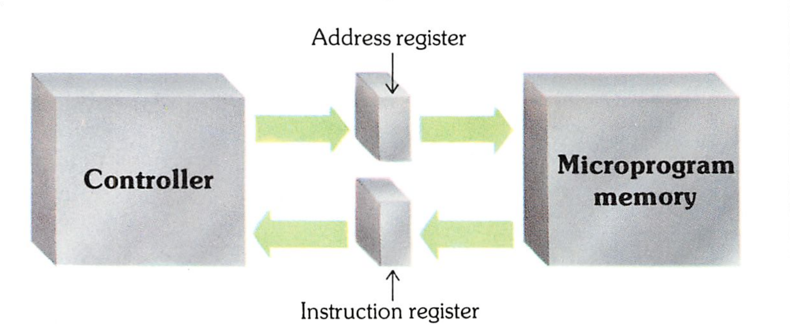
DIGITAL ELECTRONICS

RAM and ROM

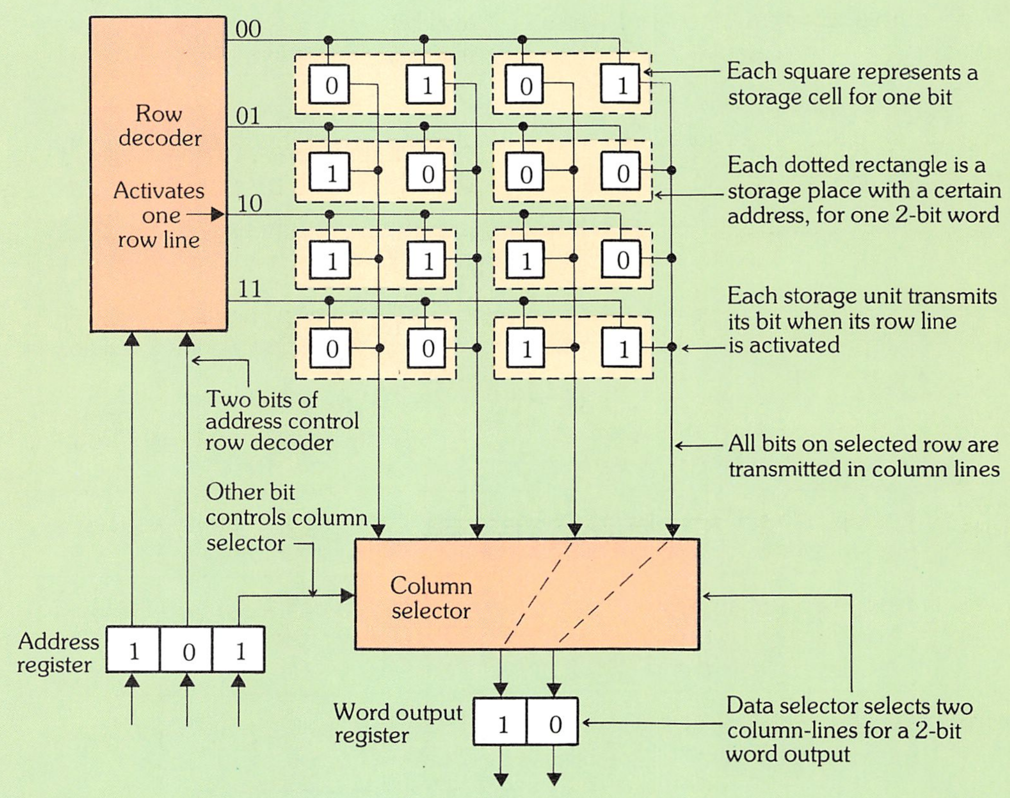
In *table 1* of *Digital Electronics 16* we saw how semiconductor memory could be classified according to storage method and access method. Having looked at serial access semiconductor memory in the form of static and dynamic shift registers, it is now time to examine the other type of semiconductor memory, **direct access memory** (sometimes called random access memory). As we said in the previous chapter, these are memories that enable data to be read or stored directly. Access times are significantly faster than for serial access memories because they don't need to be 'read' through from the beginning.

There are two types of direct access memory: **RAM** (read/write direct access memory) which can have information written to and read from it; and **ROM** (read only direct access memory) which can only be read from.

1



2


00
01
10
11
Row decoder
Activates one row line
Each square represents a storage cell for one bit
Each dotted rectangle is a storage place with a certain address, for one 2-bit word
Each storage unit transmits its bit when its row line is activated
Two bits of address control row decoder
All bits on selected row are transmitted in column lines
Other bit controls column selector
Column selector
Address register
1 0 1
Word output register
1 0
Data selector selects two column-lines for a 2-bit word output

1. ROM microprogram memory of a calculator.

2. Typical arrangement of 1-bit ROM or RAM memory.

ROM

Think back to the microprogram memory in the calculator example of *Digital Electronics 1* (*figure 1*); the circuit which forms the microprogram memory is ROM. The controller, remember, stores the address of the desired instruction in the address register. The microprogram memory then transfers a copy of the instruction (held at the address indicated) into the instruction register. The controller then reads the instruction.

An ordinary calculator has, for example, about 256 instructions, each of which comprises 16 bits, thus making a total of 4096 stored bits of information. Each address comprises 8 bits. In order to understand how the microprogram memory works, we'll examine how the memory cells are arranged inside the device.

Direct access memory cells

Figure 2 illustrates how 1-bit ROM or RAM memory cells are typically arranged. For the sake of simplicity, this example memory stores only 16 bits in the form of eight, 2-bit words. The 1-bit memory cells illustrated are arranged in a matrix of four rows and four columns; each pair of cells that make up a word are in the same row. (By comparison, the calculator ROM that we have just discussed would have 64 rows, each storing four, 16-bit words.)

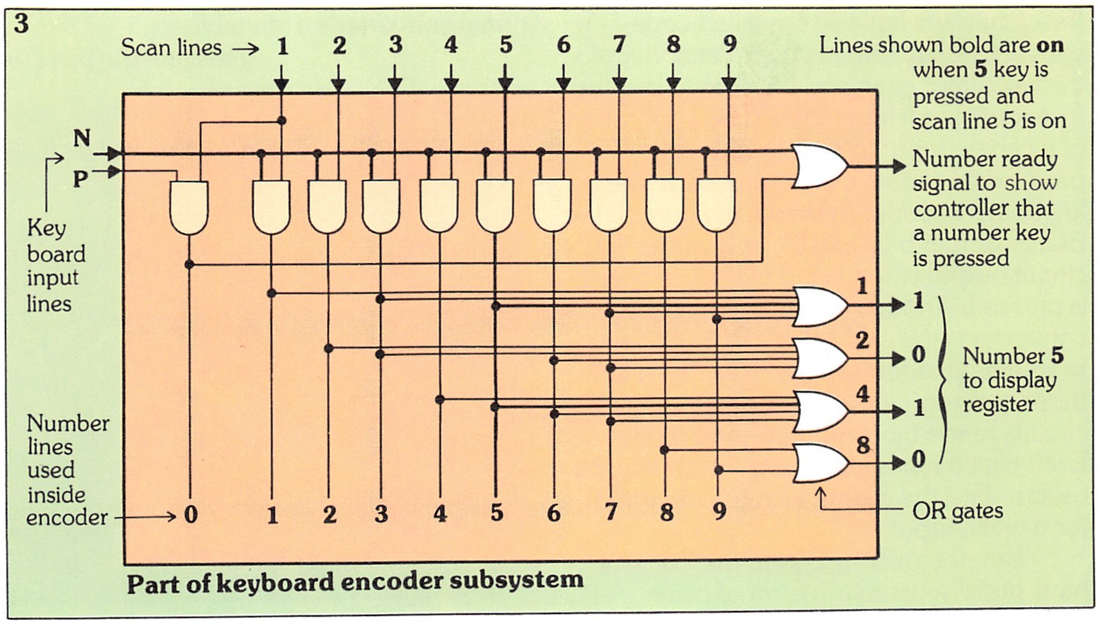
Addressing the eight words in our example requires a 3-bit address number (i.e. from zero to seven in binary). In *figure 2*, the 3-bit address number is 5, i.e. 101. The first two address bits go to a row decoder, causing it to activate one of the four row-lines – in this case row number two is addressed by the binary number 10. Once activated, the four memory cells in that row transmit the bits in their column lines. One, 2-bit word (either 11 or 10) has to be selected from these four bits – this is done by the remaining address bit. If the third address bit is 1 then the word 10 is placed in the word output register; if the remaining address bit is 0, the addressed word (11) is placed in the word output register.

Nearly all direct access type memories use a variation of this basic principle of a rectangular array of 1-bit memory cells. This enables the memory cells to be packed closely together, reducing the number of interconnections needed.

How a ROM functions

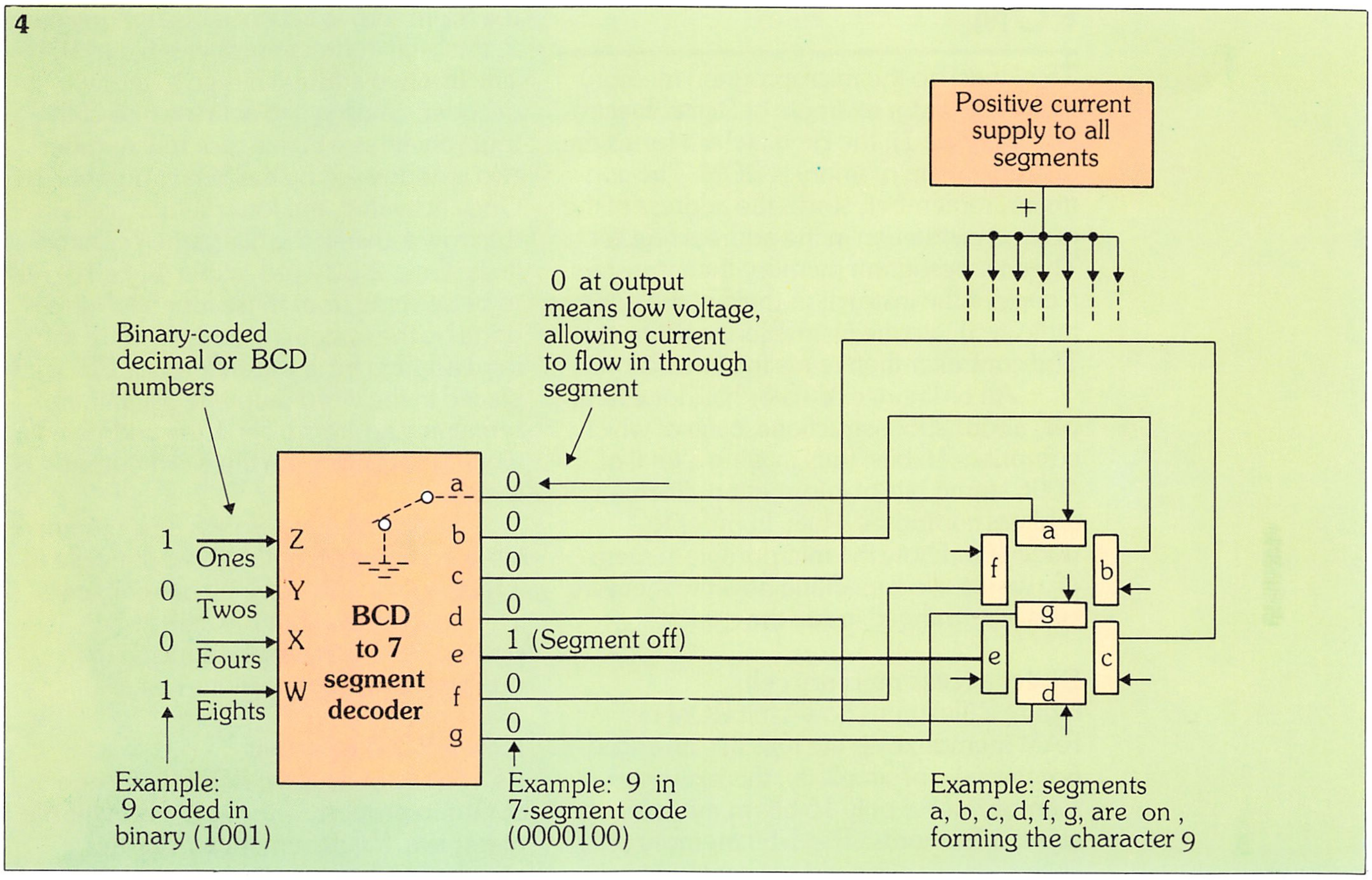
As a family of devices, ROMs have one common property – they are **non-volatile** memories. You'll remember that this means that the stored information is not lost when the power supply is turned off, and because of this their main function is the storage of frequently used program routines.

In our simple calculator example, each mathematical function would be stored in ROM; on a larger scale, ROMs


3
Scan lines → 1 2 3 4 5 6 7 8 9
Lines shown bold are on when 5 key is pressed and scan line 5 is on
N
P
Key board input lines
Number ready signal to show controller that a number key is pressed
1
2
4
8
1
0
1
0
Number 5 to display register
Number lines used inside encoder → 0 1 2 3 4 5 6 7 8 9
OR gates
Part of keyboard encoder subsystem

3. Part of a keyboard encoder subsystem of a calculator.

4


Positive current supply to all segments
+
0 at output means low voltage, allowing current to flow in through segment
Binary-coded decimal or BCD numbers
1 Ones
0 Twos
0 Fours
1 Eights
Z
Y
X
W
BCD to 7 segment decoder
a
b
c
d
e
f
g
0
0
0
0
1 (Segment off)
0
0
Example: 9 coded in binary (1001)
Example: 9 in 7-segment code (0000100)
Example: segments a, b, c, d, f, g, are on, forming the character 9

form the heart of microprocessors – storing the operating system and the data processing microprograms. ROMs are also widely used in many other branches of electronics.

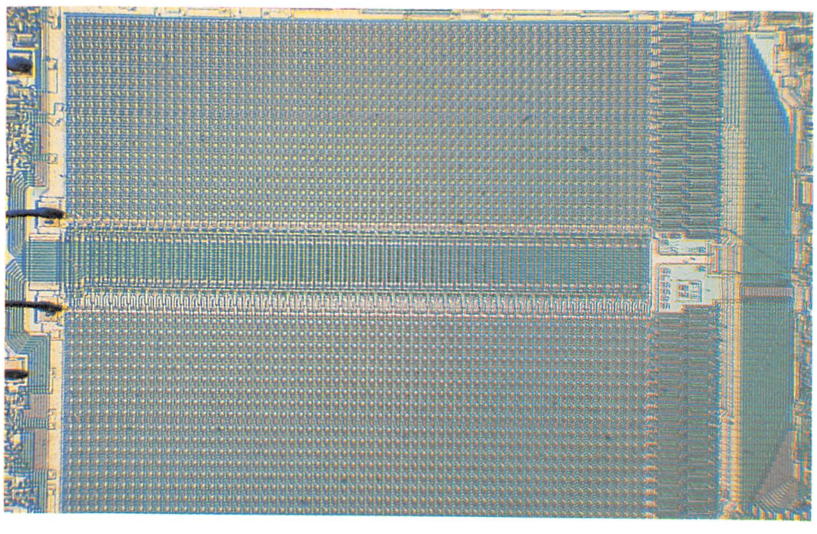
The main reason behind the usefulness and high potential of ROM is that, in practice, it acts as a code converter. Now, the circuits we have so far called code converters, say, the keyboard encoder of the calculator example, have been built up from individual logic gates. The application of a certain input to a code converter produces a certain output from it. In the keyboard encoder (redrawn in *figure 3*) the BCD number 5 (i.e. 0101) is given as the circuit output when key 5 of the keyboard is pressed. Another example of a code converter is the calculator's BCD-to-seven-segment decoder (see *figure 4*). Both of these code converters are combinational circuits made by combining logic gates. Each circuit has a truth table which may be used to find the output which is obtained for a given input.

But, the microprogram memory we have just discussed *is a form of code converter* – the address of the desired instruction stored in the memory is the code converter's input, and the instruction itself (stored at that address) is the output. We may even draw up a truth table of inputs and outputs of the microprogram memory, which corresponds to a combinational converter's truth table.

(continued in part 20)

4. A calculator's BCD-to-seven-segment decoder is an example of a code converter.

Below: photomicrograph of a ROM chip.


Neville Miles